THESIS
2

MICHIGAN STATE UNIVERSITY LIBRARIES
3 1293 01417 2674

This is to certify that the

dissertation entitled

Processor Management in 2-D Mesh Wormhole-routed Multicomputers

presented by

Dugki Min

has been accepted towards fulfillment
of the requirements for

Ph D. degree in Computer Science

Matt W. Mutka
Major professor

Date May 2, 1995

MSU is an Affirmative Action/Equal Opportunity Institution

O-12771

LIBRARY
Michigan State
University

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.

DATE DUE	DATE DUE	DATE DUE
NOV 2 4 1998		
FEB 2 8 1999		

MSU is An Affirmative Action/Equal Opportunity Institution
c:\circ\datedue.pm3-p.1

PROCESSOR MANAGEMENT IN 2-D MESH WORMHOLE-ROUTED MULTICOMPUTERS

By

Dugki Min

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DEGREE OF PHILOSOHPY

Department of Computer Science

1995

ABSTRACT

PROCESSOR MANAGEMENT IN 2-D MESH WORMHOLE-ROUTED MULTICOMPUTERS

By

Dugki Min

Processor management is one of the important services provided by the operating systems of multicomputers that serve multiple jobs simultaneously. This thesis investigates several fundamental issues that are important when designing processor management schemes for 2-D mesh wormhole-routed multicomputers. This research investigates the effects of job interactions and addresses the performance degradation due to network contentions when jobs interact. Network contentions may be crucial to the design of processor allocation strategies that allocates processors to a job from geometrically dispersed regions. A general contention model is proposed to study the effects of competing paths on network contention due to the nature of wormhole routing networks when several paths are overlapped that have different communication rates. Based on the proposed contention model, we derive analytic expressions that predict the performance detrimental effects of job interactions in terms of contention delay. The analytic study of network performance leads us to investigate the principles that may be applied when developing a scattered allocation strategy for a 2-D mesh multicomputers. Efficient methods of partitioning and

placing jobs are provided with regard to several communication parameters. We also investigate the performance effects of irregularity of job shape and size by examining a dynamic scheduling system that schedules jobs of various shapes from regular shape to irregular shape. This research outlines an approach for restricting incoming job request to use partitions from a multicomputer so that the performance advantage of regular-shaped partition can be utilized. In addition, we propose a new job scheduling discipline that can achieve low job turnaround time and high system utilization while not inappropriately favoring small jobs to the detriment of large jobs. The discipline adapts its scheduling policy to the changes of workload so that it behaves in a FCFS manner under low loaded conditions, but exploits performance enhancing features of multiple queue schemes under highly loaded conditions.

Copyright © by

Dugki Min

1995

To my parents and my wife

ACKNOWLEDGMENTS

I would like to express my sincere thanks to my advisor, Professor Matt W. Mutka, who has guided me throughout the academic and research years of my Ph.D. program. His consistent guidance and patient encouragement has helped me mature as a researcher. I am grateful for many discussions and invaluable comments he has provided.

I am also very grateful to the other members of my dissertation committee: Professor Lionel M. Ni, for his wise and invaluable comments and criticisms; Professor Moon Jung Chung, for his friendly advice and careful reviewing of the manuscript; Professor James H. Stapleton, for his willingness to answer my questions.

I would like to take this opportunity to thank those who have helped me during my stay at Michigan State University. Many thanks to Professor Rawle Hollingsworth for his willingness to help and support me financially. I thank all the brothers and sisters in the New Hope Baptist Church. In particular, I am grateful to Dr. Chang, Dr. Han and Pastor Cho for their sincere care, love and prayer.

I am indebted to my family members. I express my hearty thanks to my parents for their constant love and encouragement that have helped me do my best. Many thanks to my father-in-law and mother-in-law for their understanding and encouragement. I also thank my sisters, brothers-in-law and sisters-in-law for their encouragement and support to make decisions. Finally, my very special thanks to my wife, Eunmi Choi, for her continuous patience, love and prayer throughout my graduate studies. She spent many sleepless nights with me in discussing on research issues.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Parallel computing systems in the form of massively parallel computers (MPC) have become popular for researchers in many scientific fields. Researchers map scientific computations to parallel programs in an attempt to utilize all the processors of an MPC effectively. Some computing problems may be able to efficiently utilize all the processors and memory of an MPC and nearly achieve linear speedups in execution times. Many other computing problems, however, can only efficiently execute on a smaller subset of the processors. Due to synchronization and communication overhead, the computing problems will not improve in performance as the number of processors allocated to the problems increases. Therefore, in order to utilize the processors in an MPC efficiently, multiple jobs must be allocated to the MPC simultaneously. The multi-user environment greatly complicates the processor management task of the operating system.

The portion of the operating system for processor management, called the *processor manager*, plays the role of allocating the incoming jobs generated by concurrent users to a limited number of processors. The processor manager is composed of three components. When incoming jobs arrive at the system, the first component, called the *task assigner*, characterizes the request of the jobs by determining the sizes and shapes of the subpartitions that can accommodate the incoming jobs. The jobs are

passed to the second component. The second component, called *job scheduler*, has one or more queues that are dedicated for job scheduling. When the job scheduler receives a job request from the task assigner, it places the job in one of the scheduling queues. The job scheduler schedules the jobs in the queues according to its job scheduling discipline and the characteristics of the jobs, until the scheduling is blocked by a job that cannot be allocated immediately. When a job is scheduled by the job scheduler, the third component, called the *processor allocator*, checks the status of the current system and the possibility that the job request can be allocated. If possible, the processor allocator allocates the job to the processors that are determined by its processor allocation strategy.

We concentrate on the aspects of the processor management problem in an MPC in which the processors are interconnected in a 2-D mesh topology and communicate by message passing that is based upon wormhole routing. The regular and simple structure of the 2-D mesh multicomputer can be implemented at a low cost, while many algorithms implemented for the structure exhibit good performance. In addition, the performance of the system in terms of the average network latency, throughput, saturation throughput, and hot-spot throughput is better than the other k-ary n-cube networks that have higher dimensions when the bisection width of each network is held as a constant [1].

Two types of processor management scheme have been studied for 2-D mesh wormhole-routed MPC in commercial and research fields, depending on the type of the employed processor allocation strategy. One management scheme, which has been employed in the Intel Paragon [2], allows processors from geometrically dispersed regions of the MPC to be allocated to a user's job. Although a user's job may be represented logically as a particular geometrical shape, such as a rectangular mesh, each processor of the logical mesh can be *scattered* across the 2-D mesh parallel processor when the job is allocated to the MPC. One benefit of a scattered processor

passed to the second component. The second component, called *job scheduler*, has one or more queues that are dedicated for job scheduling. When the job scheduler receives a job request from the task assigner, it places the job in one of the scheduling queues. The job scheduler schedules the jobs in the queues according to its job scheduling discipline and the characteristics of the jobs, until the scheduling is blocked by a job that cannot be allocated immediately. When a job is scheduled by the job scheduler, the third component, called the *processor allocator*, checks the status of the current system and the possibility that the job request can be allocated. If possible, the processor allocator allocates the job to the processors that are determined by its processor allocation strategy.

We concentrate on the aspects of the processor management problem in an MPC in which the processors are interconnected in a 2-D mesh topology and communicate by message passing that is based upon wormhole routing. The regular and simple structure of the 2-D mesh multicomputer can be implemented at a low cost, while many algorithms implemented for the structure exhibit good performance. In addition, the performance of the system in terms of the average network latency, throughput, saturation throughput, and hot-spot throughput is better than the other k-ary n-cube networks that have higher dimensions when the bisection width of each network is held as a constant [1].

Two types of processor management scheme have been studied for 2-D mesh wormhole-routed MPC in commercial and research fields, depending on the type of the employed processor allocation strategy. One management scheme, which has been employed in the Intel Paragon [2], allows processors from geometrically dispersed regions of the MPC to be allocated to a user's job. Although a user's job may be represented logically as a particular geometrical shape, such as a rectangular mesh, each processor of the logical mesh can be *scattered* across the 2-D mesh parallel processor when the job is allocated to the MPC. One benefit of a scattered processor

allocation strategy is that it enables a system to maximize the utilization of the MPC. Since a job can be allocated processors from any region of the MPC, processors will not remain unused simply because there is not a large enough contiguous region of processors to allocate to a job. Nevertheless, when jobs from independent users are scattered across an MPC, the communication generated by one job may negatively affect the communication delay suffered by a second job. This is because wormhole routing [3] is the communication technique used by many MPCs. The communication delay overhead is due to the sharing of network resources such as routers and wires between interleaved and independent jobs.

An alternative approach for allocating processors to independent jobs would be to require *contiguous allocations*. This means that a user's job is mapped to an MPC in a manner such that the processors allocated to one job will not have communication paths overlapped with the communication paths of other jobs. This may be feasible in an MPC with respect to the processors allocated to the jobs, but two problems occur. First, contiguous allocation can lead to a significant amount of fragmentation of processing capacity of the MPC [4, 5, 6]. Fragmentation is the unused processors of the MPC that do not form contiguous regions that are large enough to serve users' job requests. Second, communication contention between independent jobs are likely to occur even if processors are allocated in contiguous regions. This is because jobs generate I/O requests to shared disks and other devices that are attached to servers located at specific locations in the MPC. For example, the servers in the Intel Paragon are located at the edges of the 2-D mesh. Therefore, communication contention between jobs will occur when the jobs generate message traffic to the I/O servers. The contention for I/O servers means that one job can affect the performance of a second job.

1.1 Network Contention Issues

We study network contention issues that may be critical to the performance of scattered management schemes. Because the network contention causes the degradation effect on the network performance, the locations of the processors for jobs should be carefully determined in order to minimize the negative effect.

Under the current release of the Paragon operating system, the performance degradation due to interference between the competing paths has measurable effects only for messages of large size according to the results in [7]. This is because the speed of message delivery supported by the operating system is much slower than the network capacity. However, the performance impact due to network contention is expected to become more significant under future operating systems. If an operating system could deliver messages as fast as the network capacity, the negative effects of network contention increase significantly as the number of competing paths increases. We believe that future MPCs will have operating systems that can deliver messages at the bandwidth of the network.

We analyze the performance degradation due to network contention. We propose a general contention model that is suitable for analyzing the performance degradation in wormhole-routed multicomputer systems [8]. This model is a representation of arbitrarily overlapped communication paths of jobs. It has been developed by considering the network contention problem occurring while multiple jobs are allocated to the system. Based on the general contention model, we predict the contention delay due to job interactions in a 2-D mesh wormhole-routed MPC [8, 9]. If a job is allocated several processors, the internal communication of a job may use paths that overlap with the communication paths occurring within other jobs. The detrimental effect of contention caused by interference between jobs has led us to study whether it is necessary to eliminate inter-job contentions by requiring processor allocations in

contiguous regions of the multicomputer.

Based on this analysis of network performance, we investigate the principles that may be applied when developing a scattered allocation strategy for a 2-D mesh multicomputers. By isolating each communication parameter, such as the communication rate, we study whether the method of partitioning and placing jobs can change the negative effects of job interactions [10]. For the study, we draw conclusions of how to place and partition jobs in a 2-D mesh system.

1.2 System Fragmentation Issues

As jobs are allocated, the system may become fragmented. External fragmentation occurs if a contiguous partition of processors is not available to serve a job even if the needed number of processors are available. Internal fragmentation occurs if more processors are allocated to a job than is required for the job's execution.

Most research for hypercube multicomputers has focused on developing innovative processor allocation strategies that have better ability to recognize subpartitions [11, 12, 13, 14]. Due to the special high-dimensional structure of the hypercube, a hypercube system has a structural advantage that can embed many other structures into it, but this characteristic of high dimensionality makes it difficult to detect an available subcube. Compared with the hypercube, the 2-D mesh topology is simple and straightforward to detect an available submesh. A processor allocation strategy that has complete recognition ability can be developed easily. In general 2-D mesh systems, however, system fragmentation can be significantly large even though the employed processor allocation strategy has the ability of complete recognition. Incoming jobs on a general mesh system could request computing nodes that form irregular-shaped submeshes, making the unallocated parts of the system to be irregular. Therefore, in developing a processor management strategies for a general 2-D

mesh multicomputers, the inherent property of irregularity in the size of job request and the irregularity in the shape of processor allocation should be dealt with in order to reduce system fragmentation.

We study the performance effects of irregularity of job shape and size on the performance of processor management strategies. We examine the performance effect of irregularity by examining a dynamic scheduling system that schedules jobs with requests that range from regular-shaped partitions to irregular-shaped partition. The research outlines an approach for restricting incoming job request to use partitions from a multicomputer so that the performance advantage of regular-shaped partition is utilized.

In addition, we develop a job scheduling scheme that can achieve significant performance gains by reducing system fragmentation. Since most processor management schemes have concentrated on approaches for processor allocation, the schemes have used First-Come-First-Serve (FCFS) as the job scheduling discipline. However, it has been previously established that job scheduling algorithms for parallel computing systems can have a large impact on the system utilization and job response time [15]. Schemes that use multiple queues, which reorder the sequence of jobs allocated to the parallel system, can be very effective in improving the system performance. However, such non-FCFS schemes have been criticized because they provide improved average performance by favoring small jobs at the expense of large jobs. In order to achieve improved performance by means of multiple queue job scheduling schemes without sacrificing the fairness of FCFS, we propose a new job scheduling discipline that behaves in a FCFS manner under low loaded conditions, but exploits performance enhancing features of multiple queue schemes under highly loaded conditions. The scheme does not inappropriately discriminate against large jobs.

1.3 Thesis Outline

The thesis is organized as follows. The next chapter presents a brief overview of research issues and related work on the processor management problem for different architectural platforms.

Our research related to the network contention issue is given in Chapter 3–Chapter 5. Chapter 3 describes our model of a 2-D mesh wormhole-routed multicomputer system and proposes the general contention model, called the *heterogeneous multipath contention model*, for analysis. In Chapter 4 we develop expressions for predicting the contention delay of a job. The detrimental effect of contention caused by interference within a job has led us to analyze two different kinds of communication contention. Chapter 5 we analyze the degradation of communication performance due to multiple interacting jobs in the heterogeneous multipath contention model. A divide-and-conquer strategy divides the problem into several manageable problems of computing the contention delay for the heterogeneous 2-path contention model.

The system fragmentation issue is studied in Chapter 7 and Chapter 8. The effect of job size irregularity is studied in Chapter 7 with regard to the jobs whose requests vary from regular-shaped partitions to irregular-shaped partitions. In order to evaluate the effect of irregularity, we examine a group-based job scheduling algorithm, called *BWQ-search algorithm*, which uses multiple queues for ordering jobs to be placed on a 2-D mesh multicomputer. In Chapter 8 we propose a new job scheduling scheme, called the *HELM discipline*, that adapts its scheduling policy to the changes of workload. The HELM discipline achieves improved performance by means of multiple queue job scheduling schemes without sacrificing the fairness of FCFS.

Future work and concluding remarks are given in Chapter 9.

CHAPTER 2

RESEARCH ISSUES

The rapid progress in the evolution of multicomputer systems focuses the attention of many researchers on defining and solving new problems. Many innovative strategies for job scheduling and processor allocation have been proposed and compared for several different architectures, applications, and performance requirements. An interesting fact is that the foci of the studies have changed depending on the type of system architecture.

MPC systems have been classified as *shared memory systems* and *distributed memory systems* depending on how processors and memory modules are connected. Shared memory systems have a global memory shared by all processors. The global memory reduces the difficulty of programming. Small or medium scale products are implemented commercially and in research environment [16, 17, 18, 19, 20, 21]. which include Sequent's Balance 8000, Encore's Multimax, CRAY-X/MP, BBN's Butterfly, Stanford Dash and KSR1. In distributed memory systems, called *multicomputers*, each processor has its own memory and can only access its own private memory. Communications between processors are done by passing messages through an direct interconnection network. Medium and large scale distributed memory systems have been developed commercially and in research environment [22, 23, 24, 25, 26, 27, 28, 29, 30, 31], which include NCUBE family, Intel

Paragon, Connection Machine CM-5, IBM's SP2, MIT J-Machine, Tera Computer System, Cray T3D, and NEC Cenju-3.

This chapter presents a overview of research issues and related work on the processor management problem on the shared memory and distributed memory system architectures.

2.1 Processor Sharing vs. Processor Partitioning

Processors in the shared memory systems have been treated in the literature, as if they are elements in a processor pool [32, 33, 34, 35]. In a processor pool processors can be shared by several ready users (called *processor sharing*) or can be exclusively dedicated to an assigned user (called *processor partitioning*). A research issue is the design tradeoffs between processor sharing, which may increase processor utilization at the cost of context switching, and processor partitioning, in which job turnaround time may be lower than in processor sharing.

Processor sharing scheme can be employed in the situation that the number of processors is insufficient. In that situation the processor manager should utilize processors efficiently by sharing a processor among several processes. Two approaches of processor sharing have been studied. *Dynamic allocation* is one approach in which processors are dynamically allocated to jobs on demand [32, 35]. It assumes that the number of processors available to each job may vary during the execution in a way that reflects the dynamic parallelism of the job. If some of the allocated processors are not needed temporarily, then the processors are allowed to be reallocated to other waiting jobs at the cost of context switching. Another possible processor sharing approach is a *static allocation* strategy that allocates the processors by 'time slicing'. Round-robin is an example. Even though the allocated processors are shared by other jobs, the number of processors allocated to each job is fixed during its entire execu-

tion. A dynamic allocation strategy has been compared with several static allocation strategies in [32].

Processor partitioning scheme is appropriate for the systems that have a sufficient number of processors. In the systems, processors can be partitioned according to the requests of each job and dedicated to the job during the entire execution time. 'Run-to-completion' static allocation is a good processor partitioning strategy. This kind of processor partitioning is called *processor clustering*, since a number of processors, called *clusters*, are partitioned from the processor pool and dedicated to the job during the entire execution time. The BBN's Butterfly multiprocessor is an example using processor clustering [16].

2.2 Basic Principles For Job Scheduling

In a processor pool model, a job can be allocated in any subpartition of the system without any difference in communication performance. Therefore, the processor allocation strategy for this model is simple; if there are a sufficient number of processors for the currently scheduled job, then an available subpartition is allocated to the job. Otherwise, the job should wait until the required number of processors are available. In contrast, the system performance is mainly restricted by the performance of job scheduling. Thus, it is worthwhile to know what are the basic principles that affect the performance of a job scheduling strategy.

One of factors that affect the performance of job scheduling strategies is the system workload. Based on whether a priori knowledge of the workload is given while scheduling, job scheduling strategies have been classified into *static scheduling* and *dynamic scheduling*. A static scheduling assumes that there is a given job list and that the characteristics of all jobs are known in advance. In contrast, a dynamic scheduling assumes jobs arrive according to a stochastic process and there is no a priori

knowledge of the characteristics of all jobs to be scheduled. Many researchers have proposed solutions to static and dynamic scheduling problems for general purpose multicomputer systems [36, 5, 37, 38, 39, 40] and for hard real-time systems [41, 42].

An interesting study by Majumdar, *et al.* [33] investigated the fundamental issues that are important for static and dynamic scheduling on shared memory multiprocessors. They addressed several fundamental issues that have important roles in uniprocessor systems, such as how significant the effect of the characteristics of the workload is on the performance of job scheduling strategies on multiprocessor systems and what kind of knowledge about the characteristics of workload is important for the design of scheduling strategies for a given system. Based on their abstract model of multiprocessor systems and job scheduling strategies, the research provided basic principles underlying the performance of job scheduling strategies.

Leutenegger and Vernon [34] examined various job scheduling strategies in determining which properties of a scheduling strategy are the most significant. According to their results, strategies that allocate an equal fraction of the processing power to each job perform better than strategies that allocate processing power unequally. They also claimed that for lock access synchronization, dividing processing power equally among all jobs is a more effective property of a scheduling strategy than the property of minimizing synchronization spin-waiting, unless the demand for synchronization is extremely high. Some heuristic job scheduling algorithms and their performance analysis can be found in [43, 44, 45, 46, 47].

2.3 Graph Embedding

In distributed memory systems, the *graph embedding problem* should be considered in determining the sizes and shapes of the subpartitions that can accommodate the incoming jobs. The graph embedding problem arises when the dependency structure

of a parallel algorithm differs from the processor interconnection of the system or when the number of processes generated by the algorithm exceeds the number of processors available. The graph embedding problem is to find a one-to-one mapping of a graph onto another graph which minimizes a cost function. This problem has been applied to not only the task assignment problem in a distributed and parallel processing system [48, 49, 50, 51, 52], but also many fields in computer science, such as VLSI circuit layout [53, 54, 55], simulating one data structure by another [56, 57], and simulating one parallel processing architecture by another [58, 59].

The graph embedding problem is significant because the performance of a parallel system for a job can be affected by the efficiency of the embedding. Several quality factors have been considered which measure the communication distance between two communicating processors (the dilation factor), the communication congestion in the interconnection network (the congestion factor), and the number of extra processors used in support of communication between processes (the expansion factor). A careless embedding may increase the dilation factor, the congestion factor, or the expansion factor.

2.4 Partition Recognition Ability

The hypercube network topology for distributed memory systems raises a new issue. A simple regular partitioning scheme is likely to partition the system in a particular way, such that it is difficult to recognize available partitions. Most of the research papers on the processor management problem of hypercube systems have proposed allocation strategies concerning this issue [11, 12, 13, 14].

The system partitioning scheme of hypercube systems typically is regular in order to utilize the processors efficiently since the hypercube topology itself is regular. The number of processing nodes is assumed to be 2^k, where $k \geq 0$ and the shape of

partitions are assumed to be hypercube. Thus, one research issue is how to partition the system such that a available subcube of the required size can be recognized quickly and completely. Complete recognition means that if there are unused subcubes large enough to cover the job request, the scheme should be able to recognize them.

Many researchers proposed processor allocation strategies for dynamic scheduling in hypercube multicomputers [15, 11]. The buddy strategy for memory allocation [60] is a simple approach that is implemented in the NCUBE/six multiprocessor [24]. The *Buddy strategy* is optimal for static scheduling, but shows poor recognition ability for dynamic scheduling [11, 12, 13]. A strategy using a simple gray code, called *SGC strategy*, is another simple and statically optimal strategy whose recognition ability is twice that of the buddy strategy for the dynamic scheduling problem [11]. This strategy does not have complete recognition ability. However, the recognition ability can be complete by using multiple gray codes. A strategy that has complete recognition ability by means of multiple gray codes is presented in [11].

Several other processor allocation strategies having complete recognition ability have been proposed in the literature. The *MSS strategy* [14] is one. The strategy employs the concept of a maximal set of subcubes (MSS) in order to minimize fragmentation. The *Tree-Collapsing strategy* [13] is also a processor allocation strategy that has complete recognition ability. The strategy collapses the typical binary tree representation of a hypercube successively so that the nodes, which form a subcube that are distant, are logically nearby each other for recognition. Non-cube allocation strategies have been proposed [12, 61, 62] that can allocate to a job a number of processors that is not a power of 2.

2.5 Other Techniques For Cube Allocation

Even if the subcube recognition ability is complete, the performance of processor allocation strategies can be restricted in hypercube multicomputers due to system fragmentation. Several techniques have been investigated in order to alleviate the system fragmentation in the hypercube multicomputer systems. The *scan algorithm* [15] and the *lazy scheduling algorithm* [63] tried to improve the performance by changing the order of execution. In [15], Krueger, *et al.* have compared the roles of processor allocation and job scheduling for achieving good performance on hypercube computers. They showed that the choice of the job scheduling algorithm is more important for the overall performance of the system than the choice of the processor allocation strategy in hypercube systems.

Chen and Shin [64] have examined the performance improvement to be achieved by relocating the allocated jobs to compact the system for a large free space. They proposed a task migration strategy for the gray code allocation strategy [11]. Yu and Das [65] have proposed another approach called *limit allocation* that scale down the request size of an incoming job so that it fits into a fragmented hypercube.

2.6 External Fragmentation

Processor allocation strategies for mesh multicomputers that avoid inter-job contentions have followed a generalization of the traditional binary buddy strategy for memory management [5, 66]. Using these strategies, partitions allocated to jobs have a square submesh geometry, with the lengths of the sides of the submesh equal to $2^k, k \geq 0$. This restriction of the geometry of the shapes of jobs is not appropriate when we consider a general mesh system. For a general system, jobs may request irregular-shaped partitions as well as square-shaped partitions, such that the lengths of the sides of a partition might not equal 2^k. Another drawback of a traditional

binary buddy strategy is that large internal fragmentation occurs if it is used for jobs with irregular sizes and shapes. *Internal fragmentation* is the ratio of the number of the processors that are allocated, but not used to the number of allocated processors.

A general recognition strategy is needed to identify available submeshes of arbitrary sizes at any location in a mesh. One strategy proposed is based on *frame sliding* [6]. In this strategy, a submesh is allocated that matches the shape and size requested by an incoming job. Therefore, internal fragmentation is completely avoided. As a result of its matching capability, this scheme can be used for general sizes and shapes. Ding and Bhuyan [67] improved the performance of the FS strategy by allowing the change of the orientation of incoming jobs. However, due to the irregular sizes of jobs, it is still difficult to avoid large *external fragmentation*, which is the ratio of the number of available processors to the number of processors in the system when an allocation miss occurs. Addressing the problem of the first-fit allocation nature of the FS strategy, Zhu [68] has compared a best-fit (BF) strategy based on certain heuristic with the first-fit (FF) strategy. Their results showed that neither of FF nor BF can achieve superior performance than the other at all times in a dynamic workload, and both strategies suffer external fragmentation.

By controlling the location where a submesh is allocated by the processor allocation strategy, Sharma and Pradhan [69, 70] tried to reduce the external fragmentation. Their strategy searches free submeshes on the corners of allocated submeshes along with the corners of the mesh system so as to aggregate allocated processors. This aggregative allocation clusters the allocated processors, increasing the probability of having big free partitions and of finding a sufficient submesh to accommodate an incoming job. Bhattacharya and Tsai [71] attempted to enhance the system performance by an heuristic approach that looks into the queue of waiting jobs. They argued that the heuristic performance is heavily dependent on the nature of jobs in the waiting queue, and for a dynamic workload none of the heuristics that do not use

lookahead knowledge performs superior to the others.

Recently, Liu et al. [7] proposed a non-contiguous processor allocation strategy, called the multiple buddy strategy. This strategy may allocate processors that are scattered across the parallel computing system to a job. The authors showed that the approach worked well, especially on an Intel Paragon because the communication overhead to send a message on the system was relatively large in comparison to the bandwidth of the network. Therefore, communication network contention between messages was not a problem.

2.7 Rectangular Packing Problem

The processor allocation problem in a 2-D mesh can be interpreted as a theoretical problem, called the *rectangular packing problem.* The rectangular packing problem is known to be NP-complete [72]. Researchers have provided several approximation algorithms for the problem and analyzed them theoretically.

The static processor allocation problem on mesh systems has been considered a variant of the bin packing problem [73, 74, 75]. Coffman, *et al.* [73, 74] interpreted the processor allocation problem on a 1-D mesh, *i.e.*,array, system as a two-dimensional optimal rectangle packing problem. Li and Cheng [76] applied the same analogy to the 2-D mesh allocation problem, interpreting it as a three-dimensional optimal rectangle packing problem. The packing problem has been shown to be NP-hard even for 1-D mesh, *i.e.*,array, systems that have only two processors [72]. Therefore, researchers have concentrated on providing near-optimal approximation algorithms that have polynomial execution time and a reasonably absolute and asymptotic performance bound [73, 74, 77].

A decision version of the two dimensional optimal rectangle packing problem has been studied extensively [78, 79, 76]. The decision problem is considered a formal

model of the system partitioning problem of mesh systems. NP-completeness of this decision problem has been proved with different constraints; e.g., rectangle packing concerning orientation [78] and square packing problem in which all rectangles are squares [79]. Some heuristic polynomial algorithms for the decision version of the rectangle packing problem have been proposed and analyzed [76].

2.8 Performance Studies of Wormhole Networks

Our analysis of job interactions in the following chapters is associated with the performance studies of a wormhole-routed interconnection network under contention. Many researchers have investigated the performance of wormhole-routed networks from different perspectives. Among the many important studies we limit our discussion mainly to the work that we have found to be the most related to the development of our expressions for evaluating contention.

Dally [1] analyzed latency due to contention and hot-spot throughput of k-ary n-cube communication networks for various dimensions under the assumption of constant wire bisection. He developed an estimate for contention delay that is similar to our work. By assuming e-cube routing in a k-ary n-cube network, the latency due to contention is calculated by multiplying the probability of a collision with the expected latency due to a collision along the dimensions that a message travels. At each dimension of the network, the probability that a message skips a dimension is considered since there are n dimensions the message can travel. To compute the expected waiting time of a message by collisions he developed a quadratic equation for the expected waiting time for a collision. He compared measurements from a network simulator to the latency predicted by his expression. He claimed that his simulation agrees with the prediction within a few percent until the network approaches saturation.

Another important investigation related to our study was performed by Chittor

and Enbody [80, 81, 82]. They showed that 2-D/3-D mesh networks provide much higher performance than popular hypercube networks when the effect of contention is negligible. However, in the case of large multicomputers that use mesh networks, contention for network channels can be significant. They studied the effect of contention for a given mapping of parallel tasks on a set of multicomputer nodes. A metric called the *path contention level* was introduced as a measure of contention and the quality of the mapping. They showed that the effect of contention requires an upper bound on the rate at which messages can be injected into the network by a node. For a given communication pattern and mapping they showed how to compute the saturation point. They analyzed the case of random mapping and showed that random mapping may not be advisable for large systems having hundreds or thousands of nodes that use mesh networks.

A performance study of wormhole-routed mesh networks under no contention has been studied by Adve and Vernon [83]. The authors proposed a closed queuing network that includes message pipelining and blocking and the asymmetric virtual channel. By using that model, they examined the performance and the scalability of 2-D networks in which nodes can make multiple requests before blocking for responses, as well as for traffic patterns that exhibit nearest-neighbor communication locality. Seth [84] stochastically evaluated the performance of multicomputers with several mesh network topologies and switching techniques when the channel width is constrained.

In order to improve the performance of the wormhole networks under contention situation, several techniques has been proposed. First, the packetization technique has been investigated [85]. Packetization breaks long messages into a set of smaller messages, each of which is transmitted separately. This techniques has advantages of increasing throughput and a better distribution of message latencies, but the overheads of message fragmentation and reassembly are large. Next, to increase the

network throughput and to avoid the deadlock a technique has been employed that divides the buffer storage associated with a network channel into several virtual channels [86, 87]. Virtual channels decouple allocation of buffers from allocation of channels by providing multiple buffers for each channel in the network. The performance of wormhole networks using virtual channels has analyzed in [88]. Finally, adaptive routing can adapt to dynamic changes of network conditions, such as congestion or the fault of a node [89]. Therefore, a system using an adaptive routing has advantages of taking another path the following conditions. While message traffic is heavy on one path, message latency can be reduced by sending messages along the alternate paths. While a faulty node exists on one path, communication is possible by sending messages along the alternate paths. Several adaptive routing strategies have been proposed in [90, 91, 92]. Kim and Chien [93] investigated the effect of all the above techniques on the performance of wormhole-routed networks under a workload of a mix of short and long messages. They used an M/G/1 queuing model to explore the performance effect of message size in wormhole routing networks.

CHAPTER 3

A PERFORMANCE STUDY OF WORMHOLE NETWORK

The communication performance of an interconnection network depends on the switching technology of the network. In the switching technique called *wormhole routing*, which is employed in most currently implemented MPCs, a message's transmission time is relatively independent of the distance that the message travels under contention-free conditions [1]. This means that assigning computing nodes to a job that are scattered across an MPC does not increase the transmission time. However, contention for the communication bandwidth may significantly increase the delay of sending messages.

An important research issue of allocating processors in wormhole-routed 2-D mesh multicomputer systems is to characterize the contentions caused by interactions between jobs. A performance study of the interactions between jobs can be reduced to a performance study of the interactions between the communication paths of communicating processes that share channels. This is an analysis of the effect of the communication of one path upon the performance of another path due to wormhole routing. If communication paths pass through the same channel, then the communication time experienced by a message on a path can increase due to contention. The

amount of contention depends on the communication traffic rates and the amount of overlap in the communication paths.

This chapter describes the system and analytic models for a performance study to analyze the contention delays that occur in contiguous and scattered processor allocations. The goal of the performance study and our approaches to achieve the goal are presented as well. The actual analysis of contention delays is presented in the following two chapters.

3.1 System Model

A job considered in this chapter is composed of several parallel processes which communicate with other processes in a logical communication pattern. Each process is allocated to a processor of the 2-D mesh multicomputer. The 2-D mesh system has a large number of processing nodes, each node containing a processor, a memory, and a router. The nodes are interconnected by bidirectional channels in the form of a 2-D mesh or a 2-D torus. The processors communicate by passing messages over the interconnection network. We assume that a message is a packet, and the terms are used interchangeably.

In the interconnection network, wormhole routing transmits messages between processors assigned to each job. A survey of routing techniques for wormhole networks has been presented by Ni and McKinley [89]. Each message in a wormhole network is composed of a number of flow control digits called *flits*. The header flit controls the route of the message and the remaining flits follow the header flit in a pipelined fashion. Once a channel has been acquired by a message, it is reserved for the message. The channel is released when the last flit of the message is transmitted on the channel. If the header flit encounters a channel in use by some other message, the header flit is blocked until the channel is released. When a message is blocked, the remaining flits

in the pipeline stay in flit buffers along the route. Messages between processors in our model use a deterministic routing scheme known as XY routing [25, 22]. The XY routing algorithm sends packets on the X-dimension first and then the Y-dimension. There are no virtual channels implemented in the model.

The time required to send a message from one node to another through the interconnection network is composed of transmission delay and routing delay. Transmission delay determines the lower bound of the communication delay and is dependent on the switching technology used. For wormhole routing, the transmission time is $k_0 + k_1 * Dist + k_2 * (L - 1)$, where $Dist$ is path length, L is message length and k_0, k_1 and k_2 are system dependent constants. The first term, k_1, represents the fixed network overhead. The second term, $k_1 * Dist$, represents the time for the header flit to set up a path and the last term, $k_2 * (L - 1)$, is the time to pass a message after a dedicated path is established. Examples of the constant values used in the current 2-D mesh wormhole-routed multicomputer systems can be found in [94, 95].

Routing delay is divided into two parts: queuing delay and contention delay. *Queuing delay* is caused by the stochastic arrival pattern of messages to a communication system that can only serve the messages at a fixed rate. This delay dominates the communication time as the average rate of message arrivals to the communication system becomes large. To remove this delay in our model, the rate of message generation is restricted by a source node. That is, the source node of a message does its own local computation after the message has been sent to the destination, and the computation takes at least as long as the message delay to the destination. The idea of local computation in injecting messages has been employed in other work [96, 81]. Recent work in [97] suggests that this idea of limiting injection rate can actually improve the performance of network. The average time of local computations by a source node is called *Mean Time Between Sending messages* or MTBS. The inverse of MTBS is called the *mean communication rate.* At the destination node, packet

arrivals are taken from the network without any waiting time.

The second source of routing delay is contention. *Contention delay* occurs when messages of two or more communication paths attempt to use a single channel at the same time. Contentions are classified into two classes: internal contention and external contention. *Internal contention* occurs when two or more routing paths within a job try to use a physical channel at the same time. This type of contention is caused by the rate and pattern of the communication of a job and therefore it is an inherent property of each job. It can occur in both the contiguous and scattered allocation models. *External contention* occurs when two or more routing paths of different jobs try to use the same physical channel at the same time. When the wormhole routing mechanism is used with the scattered allocation model, external contentions cause additional delays in communication time.

For convenience, we call the contention delay seen by a message due to only internal contentions the *internal contention delay* of the message. The contention delay due to a combination of internal contentions and external contentions is called the *(general) contention delay* of the message. The internal contention delay added to the queuing delay and the transmission delay is called the *internal communication delay* of the message. The communication delay that is computed by using the general contention delay instead of the internal contention delay is called the *(general) communication delay* of the message.

3.2 Analytic Model: The Multipath Contention Model

In this section, our analytic model is proposed to analyze the effect of job interactions on communication performance. Consider the example illustrated in the part (a) of Figure 3.1. It is a snapshot of a 6x10 2-D mesh system that allows scattered

allocation. Currently, three jobs are allocated to the mesh system. Job 2 is assigned to a contiguous submesh, and Job 1 and Job 3 are assigned to scattered submeshes. We are interested in how Job 2 interacts with Job 1 and Job 3 when it shares routers and wires of its allocated processing nodes. The long arrows illustrate an example of three paths sharing physical channels between the nodes. P_1, P_2, and P_3 belong to *Job* 1, *Job* 2, and *Job* 3, and the MTBSs of those paths are $mtbs_1$,$mtbs_2$, and $mtbs_3$, respectively. The part (b) of the figure redraws the paths in the form of our analytic model that has a stair-layered pattern.

A contention may occur when messages of P_1, P_2, or P_3 attempt to use a single channel at the same time. The location at which the contention may occur is called a *contention point*, and the paths that compete for a physical channel at the contention point are called the *competing paths*. The first contention point at the first communication channel that a message faces when it is injected from a processor to the network is called *starting contention point*. The remaining contention points along a communication path are called the *intermediate contention points*. For example, in the part (b), the 'O' symbols on P_2 illustrates the starting contention point due to P_3, and the 'X' symbol on P_2 illustrates the first intermediate contention point due to P_1.

Let us examine D^{P_2}, which is the communication delay of P_2. The part (c) of Figure 3.1 determines the time related to the communication delay. If there are no contentions, D^{P_2} is merely the transmission delay, which is approximately $(d2 + d3 + L) * T_t$, where T_t is the flit transmission time, $(d2 + d3)$ is the distance of message passing, and L is the number of flits in a message. If $L >> (d2 + d3)$, then the transmission delay is approximately $L * T_t$. However, P_2 has possibilities of external contention in two places. The contention delay at the first contention point of P_2 is called the starting contention delay of P_2 and is labeled C_1. The next contention may occur with P_1 where messages on P_1 are injected into the network. This contention

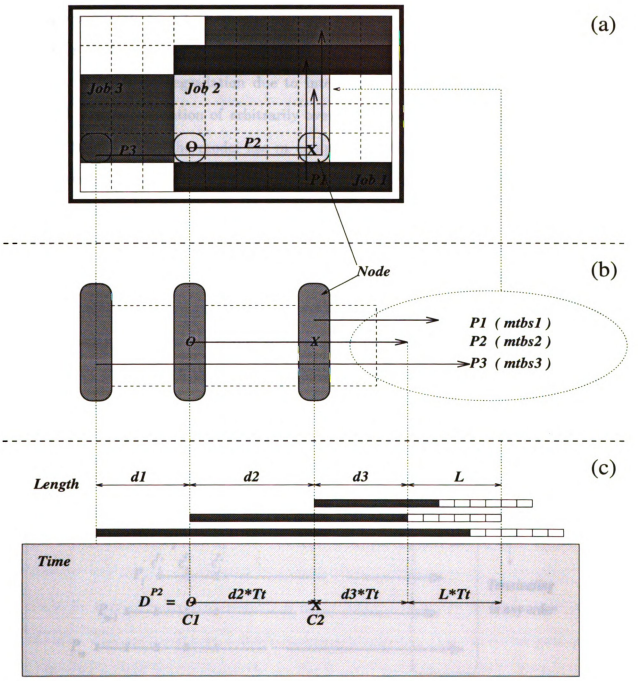


Figure 3.1. A snapshot of a 6x10 2-D mesh system with three interacting jobs; A close-up view of a 3-path contention is shown.

delay labeled C_2 is called an intermediate contention delay of P_2. The communication delay of P_2 will be the sum of the transmission delay and the contention delays, which is approximately $C_1 + C_2 + L * T_t$.

Our analysis uses a contention model called the *multipath contention model* to study performance degradation due to interactions between jobs. The contention model is a representation of arbitrarily overlapped communication paths of several jobs. Suppose that the model has m competing paths and let P_1,...,P_m be the m competing paths. The m competing paths can be rearranged by sorting in the order of the occurrence of the starting contention points such that the m-path contention model forms a stair-layered pattern, as illustrated in Figure 3.2. Since the transmission delay is fixed and relatively independent of the distance traveled by a message and the effect of queuing delay is removed in our model, the only cause of variance in the delay of a message is due to contention delays. Therefore, without loss of generality, the paths in the contention model are assumed to be sorted in the order of the occurrence of the starting contention points.

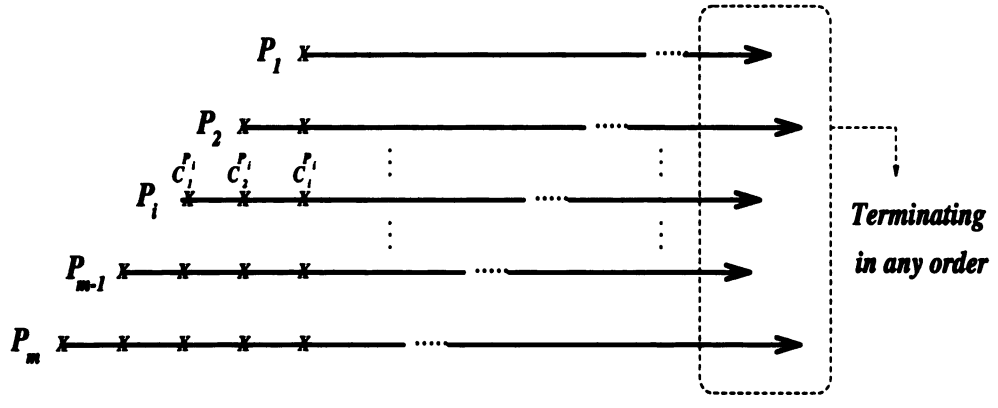


Figure 3.2. Stair-layered m-path contention model.

3.3 Goal and Approaches

Our goal is to predict D^{P_j} for all j, which is the communication delay seen by a message sent through P_j. In our model, D^{P_j} is composed of contention delay and transmission time. The contention delay of D^{P_j}, which is C^{P_j}, can be computed by accumulating $C_i^{P_j}$s for all i, which are the contention delays at the ith contention point on P_j. Note that with the exception of P_m, P_j has j contention points. Therefore, D^{P_j} can be written as follows,

$$D^{P_j} = L * T_t + \sum_{i=1}^{i=j} C_i^{P_j}$$

where T_t is the flit transmission time, L is the number of flits in a message. Since the message transmission time is a known constant when the size of the system is known, what we have to determine are $C_i^{P_j}$, for all i.

To compute $C_i^{P_j}$, two different approaches are presented in the following two chapters. The first one in Chapter 4 employs queuing theory, considering the communication in a wormhole-routed network a stochastic model. This approach is simple for deriving formulas, but it is appropriate only for the multipath contention model in which all paths have the same MTBS. Thus, this approach may be used to compute the internal contention delay of a job that is allocated in a contiguous region. In contrast, the second approach in Chapter 5 is more complex to derive formulas, but it can be applied to general contention model whose competing paths have different MTBSs. This approach uses a divide-and-conquer strategy that divide the mulitpath contention model into manageable several 2-path contention models and combines them.

CH

AN

CO

As a

when

theor

clear

by in

comm

occur

from

comm

start

to pr

that

incre

the n

start

algor

CHAPTER 4

ANALYSIS OF INTERNAL CONTENTION DELAY

As a first attempt to examine the interference and detrimental effect that can occur when multiple jobs share communication bandwidth on a MPC, we apply queuing theory to develop a set of formulas for evaluating internal contention delay of communication paths in a wormhole network. The detrimental effect of contention caused by interference between competing paths has led us to analyze two different kinds of communication contention. As described in the previous chapter, starting contention occurs when a processor attempts to access the network at the first hop on its route from the source to destination. Intermediate contention is the contention facing a communication path as the message arrives at intermediate nodes on a path. The starting contention of a path will increase if the processes on the path are allocated to processors of the multicomputer that is internal to the other communication paths that occupy surrounding processors. Conversely, the intermediate contention of a job increases if the processes of the job are allocated to nodes that are dispersed across the multicomputer and around processors allocated to other paths. The evaluation of starting and intermediate contention may provide valuable insight on how a processor allocation strategy should assign jobs to submeshes.

We develop formulas for evaluating starting and intermediate contention that a job faces without regard to the interactions with the other jobs. They are based on a stochastic model of communication in a wormhole-routed network. This formula works for a wide range of communication traffic rates, which includes rates that go beyond a saturation point. This analytic study enables us to analyze internal contention delays of a job that have a complicated communication pattern. We compare the evaluation of our formulas to a simulation of the communication network and show that our analysis yields very good results for predicting contention within wormhole networks.

The rest of this chapter is organized as follows. Section 4.1 describes the analytic model on which we analyze the contention delay within a job in Section 4.2. Concluding remarks are given in Section 4.4.

4.1 The Homogeneous Multipath Contention Model

There are two parameters in the multipath contention model that affect the utilization of a shared channel: the number of competing paths and the message communication rate. By restricting each path to have the same message communication rate, we can make a specific multipath contention model called the *homogeneous multipath contention model.* The analysis of the homogeneous mulitpath contention model provides a way to predict the internal contention delay seen by a message in a job.

Figure 4.1 illustrates an instance of the homogeneous multipath contention model that has five layers. The 'O' symbols represent starting contention points and the 'X' symbols represent intermediate contention points. To analyze the homogeneous multipath contention model, we define the following notations for convenience. The notations are illustrated by using the middle path.

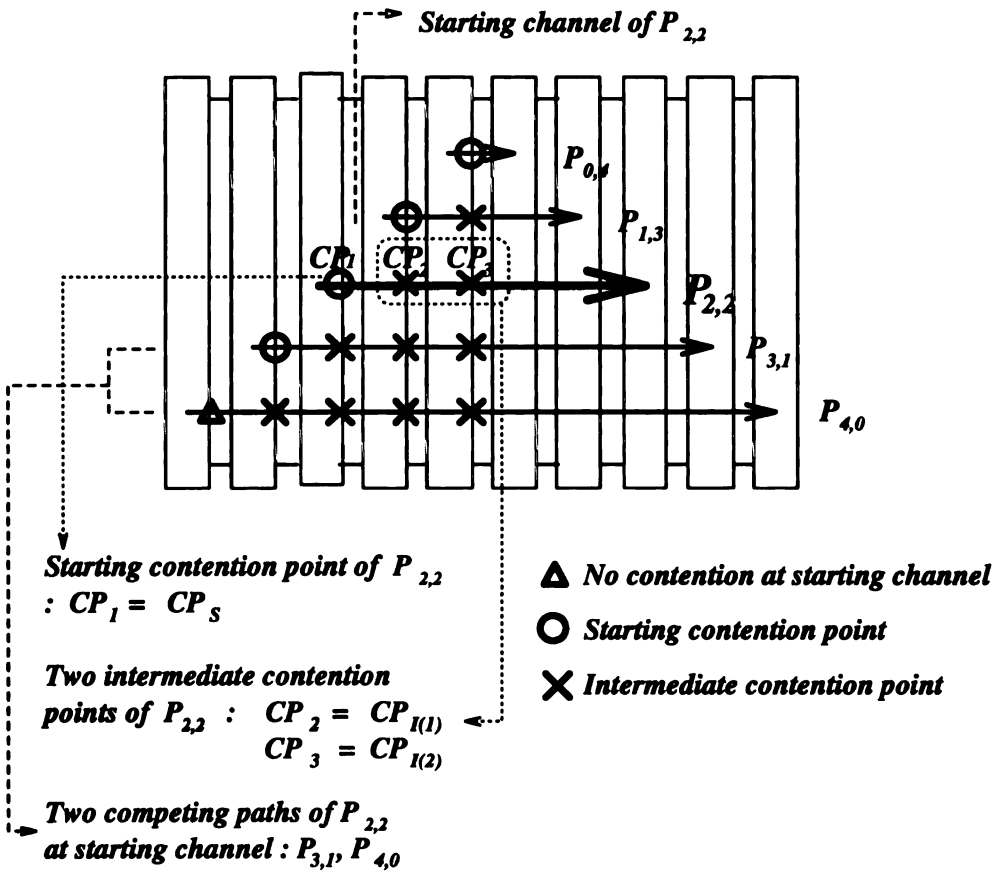


Figure 4.1. A homogeneous five-path contention model.

- $P_{u,d}$ represents a path that encounters d competing paths at its starting channel and has u intermediate contention points. For a given path P, u is the number of paths stair-layered above the path P and d is the number of paths stair-layered below the path P. For example in Figure 4.1, $P_{0,4}$ is the top path, $P_{4,0}$ is the bottom path and $P_{2,2}$ is the third path from the top.

- CP_i^P is the ith contention point on path P. Figure 4.1 shows all contention points on the path $P_{2,2}$.

- Pr_i^P is the probability that a contention occurs at CP_i^P. This is the contention probability, which is equal to the channel utilization of paths that compete with P.

- Ct_i^P is the mean contention delay at CP_i^P. $Ctmax_i^P$ represents a maximum expectation of Ct_i^P.

- $C^{P_{u,d}}$ is the mean contention delay of a message sent on path $P_{u,d}$. It is computed by adding all contention delays at each contention point on $P_{u,d}$ (i.e. $\sum_{i=1}^{i=k} Pr_i^P * Ct_i^P$). $Cmax^{P_{u,d}}$ is a maximum expectation of $C^{P_{u,d}}$. $Cmax^{P_{u,d}}$ is composed of $Cmax_S{}^{P_{u,d}}$ (the maximum expectation of the starting contention delay) and $Cmax_I{}^{P_{u,d}}$ (the maximum expectation of the intermediate contention delay).

- $D^{P_{u,d}}$ is the mean communication delay of a message sent on path $P_{u,d}$ and is the sum of the transmission delay and the contention delay, $C^{P_{u,d}}$. In our communication model, the transmission delay is a constant, $L * T_t$ where L is message length and T_t is flit transmission time. $Dmax^{P_{u,d}}$ is a maximum expectation of $D^{P_{u,d}}$.

- $MTBS$ is the mean time between the sending of a message.

- $\lambda^{P_{u,d}}$ is the mean message arrival rate of $P_{u,d}$. Note that due to the assumption of flow control, $\lambda^{P_{u,d}}$ is not constant, but a function of $D^{P_{u,d}}$. For convenience, we use $\lambda_{u,d}$.
- $\lambda_S^{P_{u,d}}$ is the mean message arrival rate at the starting channel of $P_{u,d}$.
- $\lambda_{I(k)}^{P_{u,d}}$ represents the mean message arrival rate at the kth intermediate channel of $P_{u,d}$.

4.2 Analysis

Consider a path, P, that has k contention points including the starting contention point. The expected contention delay of a message sent through path P can be computed by adding the expected contention delays at the contention points on the path. The expected contention delay at the ith contention point on P, C_i^P, is given by multiplying the probability of a contention at the point, Pr_i^P by Ct_i^P, which is the expected contention delay when the contention has occurred. The expected communication time of the path P, D^P, can be written as follows:

$$D^P = L * T_t + \sum_{i=1}^{i=k} Pr_i^P * Ct_i^P \tag{4.1}$$

The probability of contention at the ith contention point of P can be interpreted as the probability that the channel is used by the competing paths. To compute this probability, we must know the number of P's competing paths at the contention point. By observing the Figure 4.1, we recognize that the number of competing paths at the starting contention point, called the *starting contention level*, can be larger than one. The number of competing paths at any intermediate contention point, called *intermediate contention level*, is always one. Therefore, we decompose the summation that includes Ct_i^P into the starting contention delay and the intermediate contention

delay.

4.2.1 Intermediate Contention Delay

Figure 4.2 illustrates how to compute the intermediate contention probability and delay for the path, $P_{2,2}$, using the five layer communication model of Figure 4.1. First we concentrate on the second contention point of path $P_{2,2}$ which we call $CP_2^{P_{2,2}}$. Since $CP_2^{P_{2,2}}$ is the first intermediate contention point of path $P_{2,2}$, $CP_2^{P_{2,2}}$ is named $CP_{I(1)}^{P_{2,2}}$. It has only one competing path, $P_{1,3}$. To compute the contention probability at $CP_{I(1)}^{P_{2,2}}$ due to $P_{1,3}$, which is called $Pr_{I(1)}^{P_{2,2}}$, we evaluate the utilization of $CP_1^{P_{1,3}}$ due to $P_{1,3}$. We can compute $Pr_{I(1)}^{P_{2,2}}$ by approximating the utilization by referring to a simple M/M/1 queueing model that uses the channel capacity at $CP_1^{P_{1,3}}$ as the server.* Message arrivals are generated by $P_{1,3}$ as shown in the small box (a) of Figure 4.2. The mean arrival rate, $\lambda_1^{P_{1,3}}$, is the inverse of the mean inter-arrival time that depends on the mean communication time of the messages generated through $P_{1,3}$. In fact we do not know the mean communication time, $D^{P_{1,3}}$. Instead, we approximate $D^{P_{1,3}}$ to the expected maximum communication time, $Dmax^{P_{1,3}}$, as the maximum bound of the communication time. Suppose we know $Dmax^{P_{1,3}}$. Then $\lambda_1^{P_{1,3}}$ is given as follows:

$$\lambda_1^{P_{1,3}} = 1/(MTBS + Dmax^{P_{1,3}}) = \lambda_{1,3} \tag{4.2}$$

The mean service rate, $\mu_1^{P_{1,3}}$, depends on the mean service rates of the other contention points on $P_{1,3}$ due to wormhole routing. If a message header is blocked at one of the later contention points, then the entire flit stream is blocked. Therefore, the service time equals the mean communication time of $P_{1,3}$, which is $D^{P_{1,3}}$. If we use

*The queueing model is only an approximation of the system. We evaluate the quality of the approximation when we later compare it to a simulation model.

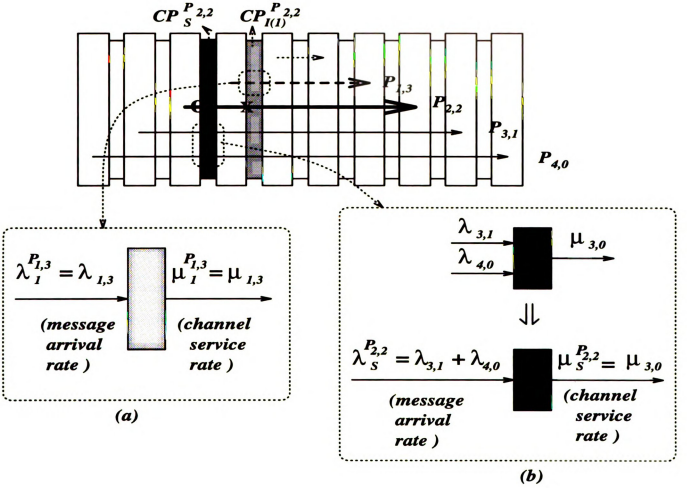


Figure 4.2. Illustration of Computation of an Intermediate and Starting Contention Probability.

$Dmax^{P_{1,3}}$ as a estimate of $D^{P_{1,3}}$, then $\mu_1^{P_{1,3}}$ is computed as

$$\mu_1^{P_{1,3}} = 1/Dmax^{P_{1,3}} = \mu_{1,3} \tag{4.3}$$

Therefore, by means of queueing theory, we can approximate the channel utilization of $CP_{I(1)}^{P_{1,3}}$ due to $P_{1,3}$ by dividing $\lambda_1^{P_{1,3}}$ by $\mu_1^{P_{1,3}}$. That is

$$Pr_{I(1)}^{P_{2,2}} = \lambda_1^{P_{1,3}}/\mu_1^{P_{1,3}} = \lambda_{1,3}/\mu_{1,3} \tag{4.4}$$

We find the probability of contention at the other intermediate contention points of path $P_{2,2}$ in a similar manner.

Next, we need to compute the expected delay at an intermediate contention point, which is the term Ct_i^P of Equation 4.1. When a contention occurs at $CP_{I(1)}^{P_{2,2}}$ with $Pr_{I(1)}^{P_{2,2}}$, the expected maximum contention delay is the expected maximum channel reservation time of the conflicting message on $P_{1,3}$. In this case, however, the channel has been reserved by the message and therefore the other competing paths fail to acquire the channel. We do not need to consider $P_{2,2}$, $P_{3,1}$ and $P_{4,0}$. Therefore, the contention delay at the first intermediate contention point on path $P_{2,2}$ is $Ct_{I(1)}^{P_{2,2}}$, which equals $Dmax^{P_{1,0}}$, the maximum delay on a path that has one competing channel above it in the stair-layered pattern and no competing paths below it. $Dmax^{P_{1,0}}$ is a value that we can compute. It is computed by adding the transmission delay of $P_{1,0}$, which is $L * T_t$, with the expected maximum contention delay of $P_{0,0}$, which is $L * T_t$. Note that when a contention occurs with the path $P_{0,4}$ at $C_1^{P_{1,3}}$, the channel is reserved for the maximum time, which is $L * T_t$.

We can generalize the computation of the expected maximum contention delay caused by u intermediate contention points on $P_{u,d}$ in a model with basic communication pattern of lv layers, where $lv = u + d + 1$. Let $Cmax_I^{P_{u,d}}$ be the sum of the

expected maximum intermediate contention delays. Then $Cmax_I^{P_{u,d}}$ is computed as follows:

$$\begin{aligned} Cmax_I^{P_{u,d}} &= \sum_{k=1}^{k=u} (Pr_{I(k)}^{P_{u,d}} * Ct_{I(k)}^{P_{u,d}}) \\ &= \sum_{k=1}^{k=u} (\lambda_{k-1,lv-k}/\mu_{k-1,lv-k}) * Dmax^{P_{k-1,0}} \\ where \quad & \lambda_{k-1,lv-k} = 1/(MTBS + Dmax^{P_{k-1,lv-k}}) \\ and \quad & \mu_{k-1,lv-k} = 1/Dmax^{P_{k-1,lv-k}} \end{aligned} \tag{4.5}$$

To verify the formula, we simulated the basic communication model of Figure 4.1 while varying the number of layers. We used the Multisim [98] simulation package for modeling wormhole-routed multicomputer networks, which is based on CSIM [99]. For our simulation study, we assumed that the communication bandwidth of the wormhole network requires 0.05 μsec to transmit each flit of a message. Since the contention-free transmission time in the wormhole network is assumed to be a constant, we set the length of a message transmitted in the network to a large value, 500 flits. This means that the transmission delay of a message is approximately 25 μsec regardless of the distance traveled. The simulation results are represented by the solid plots and the analytic results are done by the dotted plots.

Figure 4.3 compares the prediction using Equation 4.5 with the simulation results. Each plot in the figure specifies the number of levels in the stair-layered communication pattern. The vertical axis represents the communication delay and the horizontal axis represents the MTBS of messages. Our formula does a good job of predicting the maximum bound of intermediate contention delay for the entire range of MTBS, which includes the rates when saturation occurs. Note that the estimation of prediction obtained from Equation 4.5 (the plots labeled "prediction" in Figure 4.3) compares very well with the simulation results when MTBS is very large and very small. This is

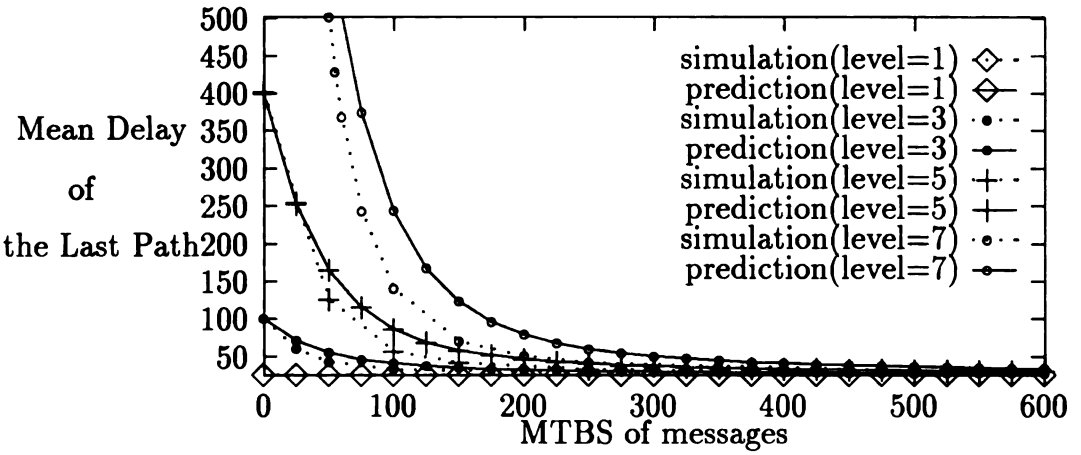


Figure 4.3. The Maximum Intermediate Contention Delay: $Cmax_I^{P_{u,d}}$.

because at these values $Dmax^{P_{k-1,lv-k}}$ is a very good estimate of $D^{P_{k-1,lv-k}}$. The estimate is not as accurate at intermediate values of MTBS, but nevertheless provides a good bound for the intermediate contention delay.

4.2.2 Starting Contention Delay

The number of competing paths at a starting channel can be larger than one. To compute the channel utilization of the starting competing paths, we cannot use the M/M/1 queueing model directly. We must use a queueing model that has multiple input sources. Assume there are n competing paths. Let S_i be the random variable of the message injection rate for ith path and let λ_{S_i} be the mean value of S_i. Then, the message injection rate at the starting channel of the path is $\sum_{i=1}^{n} S_i$. If the ith path generates messages according to a Poisson arrival process, we can prove that $\sum_{i=1}^{n} S_i$ also follows an Poisson process whose mean value is $\sum_{i=1}^{n} \lambda_{S_i}$ by using the moment generating function technique [100]. This is an approximation to reduce the multiple input queueing model to a M/M/1 queueing model. The validity of this

approximation is verified by the simulation results in Section 4.3.

Let us again consider the example illustrated in Figure 4.1. $P_{2,2}$ has two competing paths at its starting channel, $P_{3,1}$ and $P_{4,0}$, as shown in Figure 4.2. The mean arrival rate at the channel by $P_{3,1}$ is same as the mean arrival rate of $P_{3,1}$ which is $\lambda_{3,1}$. Note that all mean arrival rates at each contention point on $P_{3,1}$ are equivalent to $\lambda_{3,1}$ because the routing technology is assumed to be wormhole routed. Similarly the mean arrival rate at the channel by $P_{4,0}$ is $\lambda_{4,0}$. Therefore the combined mean arrival rate at the channel by $P_{3,1}$ and $P_{4,0}$ is

$$\lambda_S^{P_{2,2}} = \lambda_{3,1} + \lambda_{4,0} \tag{4.6}$$

where $\lambda_{u,d} = 1/(MTBS + Dmax^{P_{u,d}})$. The mean service time for a message at the channel server is

$$\mu_S^{P_{2,2}} = \mu_{3,0} \tag{4.7}$$

where $\mu_{3,0} = 1/Dmax^{P_{3,0}}$. Note that $\mu_S^{P_{2,2}}$ is neither $\mu_{3,1}$, $\mu_{4,0}$ nor a combination of two, but $\mu_{3,0}$. These rates are shown in the small box (b) of Figure 4.2. Once $\lambda_S^{P_{2,2}}$ and $\mu_S^{P_{2,2}}$ are provided, we can easily compute the contention probability at $CP_S^{P_{2,2}}$ by dividing $\lambda_1^{P_{1,3}}$ by $\mu_1^{P_{1,3}}$, that is

$$Pr_S^{P_{2,2}} = \lambda_S^{P_{2,2}}/\mu_S^{P_{2,2}} = (\lambda_{3,1} + \lambda_{4,0})/\mu_{3,0} \tag{4.8}$$

If a contention occurs at $CP_S^{P_{2,2}}$ by a message sent through one of the competing paths, the channel is reserved by the message. The expected maximum reservation time is the expected maximum communication time of a message on $P_{2,2}$ when no contention occurs at the starting channel, which is $Dmax^{P_{2,0}}$. By multiplying this delay by the contention probability, we can predict the maximum contention delay at $CP_S^{P_{2,2}}$ on $P_{2,2}$, which is $Ctmax_S^{P_{u,d}}$.

In general, the expected maximum contention delay caused by d competing paths at the starting channel of $P_{u,d}$ in the basic communication model with lv layers is $Cmax_S^{P_{u,d}}$, which is computed as follows. Note that $lv = u + d + 1$.

$$\begin{aligned} Cmax_S^{P_{u,d}} &= Pr_S^{P_{u,d}} * Ctmax_S^{P_{u,d}} \\ &= (\sum_{k=u+1}^{k=u+d} \lambda_{k,lv-k-1}/\mu_{u+1,0}) * Dmax_{u,0} \\ where \quad & \lambda_{k,lv-k-1} = 1/(MTBS + Dmax_{k,lv-k-1}) \\ & k = u+1, \ldots, u+d \\ and \quad & \mu_{u+1,0} = 1/Dmax^{P_{u+1,0}} \end{aligned} \tag{4.9}$$

It is worthwhile to explain how to solve Equations 4.5 and 4.9. Notice that Equation 4.9 uses Equation 4.5 to compute the $Dmax$'s and Equation 4.5 uses Equation 4.9 to compute the λ's and μ's. We can solve the equations by a numerical method of convergence. If we use $\lambda_{k-1,0}$ and $\mu_{k-1,0}$ instead of $\lambda_{k-1,lv-k}$ and $\mu_{k-1,lv-k}$ respectively, then the equations can be computed in backward fashion by starting the summation when k=1. By using this solution as a starting point, we can converge to the correct solution by iteratively solving the two equations based on the previous solution.

Figure 4.4 show the starting contention delays as a function of MTBS of messages for our basic communication model as the number of layers varies. The figures compare the prediction of starting contention delay due to our developed formula and the simulation results of $Cmax_S^{P_{u,d}}$. The top and bottom figures respectively show the simulation and prediction results by overlapping the lines for the model with lv = 1, 3 or 13. The trend of the plots of prediction is very similar to that of the plots of simulation for all lv and for the entire range of MTBS. In the model of thirteen levels, contention delay is very high for small values of MTBS until MTBS is less than 300. Beyond MTBS=300, the contention delay decreases slowly. This tendency gives us insight of how internal contention within a jobs effects its performance and how

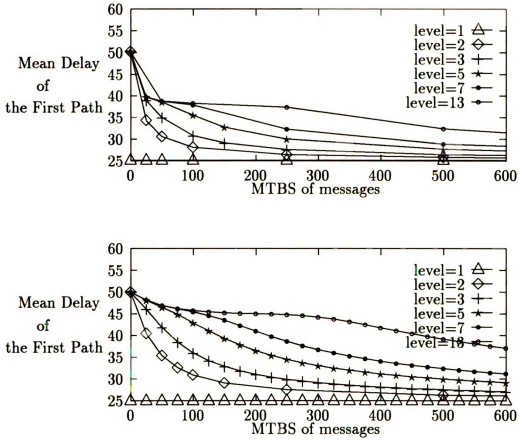


Figure 4.4. The Maximum Starting Contention Delay, $Cmax_S^{P_{u,d}}$: top - simulation results, middle - prediction, bottom - combination.

independent jobs executing on a multicomputer can affect the performance of each other. If one job uses processors that are external to the processors of another job, then the messages between processors of the external job will pass through the paths used by the internal job. The performance degradation experienced by the internal job due to the contention caused by the external job's messages can be significantly increased as the MTBS experienced by the internal job decreases.

4.3 Results

In order to verify the analysis of our homogeneous multipath contention model, we compare our analytic results with a simulation model for several configurations. For simulations, We used the same simulator as before.

4.3.1 Contention Delay of a Path

We can compute the expected maximum contention delay of a path by adding the two formulas for the starting contention delay and for the intermediate contention delay from Equations 4.5 and 4.9. Therefore,

$$\begin{aligned} Cmax^{P_{u,d}} &= Cmax_S^{P_{u,d}} + Cmax_I^{P_{u,d}} \\ &= \left(\sum_{k=u+1}^{k=u+d} \frac{\lambda_{k,lv-k-1}}{\mu_{u+1,0}}\right) * D_{u,0} + \sum_{k=1}^{k=u} \frac{\lambda_{k-1,lv-k}}{\mu_{k-1,lv-k}} * Dmax^{P_{k-1,0}} \end{aligned} \quad (4.10)$$

The average contention behavior of the basic communication pattern can be obtained if we average the expected maximum contention delays of each paths as follows:

$$Cmax^P = \frac{\sum_{k=0}^{k=lv-1} Cmax^{P_{k,lv-k-1}}}{lv} \quad (4.11)$$

Figure 4.5 shows the average behavior of our stair-layered communication pattern

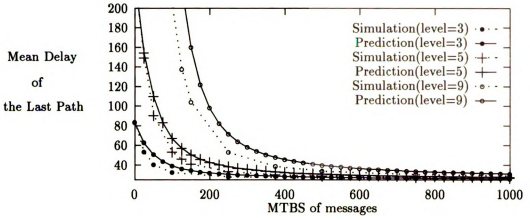


Figure 4.5. Average Contention Delay.

by varying the number of paths. Our predictions follow the simulated average contention delays over the entire range of MTBS, representing their maximum bounds. Note how the predicted values of contention delay compares with the values obtained through simulation. Equation 4.10 provides an excellent basis for evaluating contention within a job on a multicomputer using wormhole routing. It can be used as a basis for evaluating contention between independent jobs executing on a wormhole-routed multicomputer.

4.3.2 Internal Contention Delay of a Matrix Transpose Pattern

We apply our formula to a more complicated example, called the *matrix transpose communication pattern*. This communication pattern illustrated in Figure 4.6 with 4-by-4 matrix has been used by Chittor and Enbody [101]. In this pattern, a subset of nodes are actively sending messages. Only the nodes on the diagonal do not send messages. Each active node has a related node with which it exchanges messages

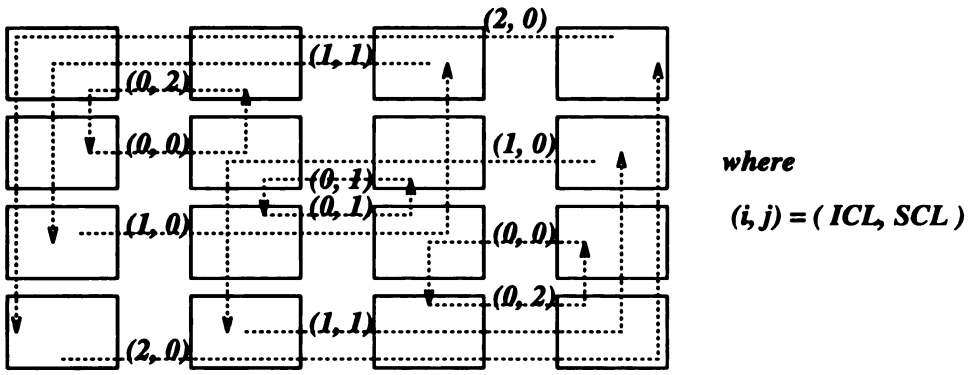


Figure 4.6. Transpose Pattern.

repeatedly. The communicating pairs of nodes using the matrix-transpose pattern are specified as

$$Node_{i,j} \longleftrightarrow Node_{j,i}$$

where $Node_{k,l}$ indicates the node in the submesh at the kth row and lth column. The specification assumes there are n rows and n columns assigned to the job, with rows and columns numbered from $0 \ldots n-1$. The symbol $\longleftrightarrow$ indicates bi-directional communication. This pattern has a high potential for communication contention. Every message generated by the job will use a path that is shared by other pairs of processors in the job. For convenience, we will refer to the matrix-transpose pattern simply as the *transpose* pattern.

To compute the expected maximum contention delay of a pattern, first we have to know the average intermediate contention level (ICL) and the average starting contention level (SCL) of the pattern. The average ICL can be calculated as follows:

$$ICL^{pattern} = (\sum_{i=1}^{i=N} ICL^{P_i})/N \tag{4.12}$$

where N is the number of paths in the pattern and P_i is the ith path. Similarly, the

average SCL of a pattern is

$$SCL^{pattern} = (\sum_{i=1}^{i=N} SCL^{P_i})/N \tag{4.13}$$

Once we evaluate the $ICL^{pattern}$ and $SCL^{pattern}$, then the expected maximum contention delay can be evaluated by using Equation 4.11. For example, the 4-by-4 transpose pattern displayed in Figure 4.6 has 12 paths. The pair of numbers at the beginning of each path indicates the ICL and SCL of the path. Therefore the average ICL and SCL of 4-by-4 transpose pattern are both 0.67 by the Equations 4.12 and 4.13.

Since this result is not an integer, the maximum expected valued of contention delay, $Cmax^{4*4transpose}$, can be calculated by an interpolation as follows:

$$\begin{aligned} Cmax^{4*4transpose} &= \sum_{k=0}^{k=0} Cmax^{P_{k,1-k-1}} \\ &+ 0.67 * (\sum_{k=0}^{k=2} Cmax^{P_{k,2-k}} - \sum_{k=0}^{k=0} Cmax^{P_{k,1-k-1}}) \end{aligned} \tag{4.14}$$

That is, we interpolate between the case when $ICL^{pattern} = SCL^{pattern} = 0$ and $ICL^{pattern} = SCL^{pattern} = 1$. Note that the basic stair-layered pattern has $SCL^{pattern} = ICL^{pattern} = 0$ when there is one layer and $SCL^{pattern} = ICL^{pattern} = 1$ when there are three layers. The term, $\sum_{k=0}^{k=0} Cmax^{P_{k,1-k-1}}$, represents $Cmax^{basicpattern}$ when $ICL = SCL = 0$, which implies that the number of layers, lv, is one. The term, $\sum_{k=0}^{k=2} Cmax^{P_{k,2-k}}$, represents $Cmax^{basicpattern}$ when $ICL = SCL = 1$, which implies lv is three.

Figure 4.7 shows an evaluation of Equation 4.14 for the transpose pattern in comparison to values of contention delay obtained by simulation. In addition, we evaluate the contention delays from our formula and simulation for transpose patterns

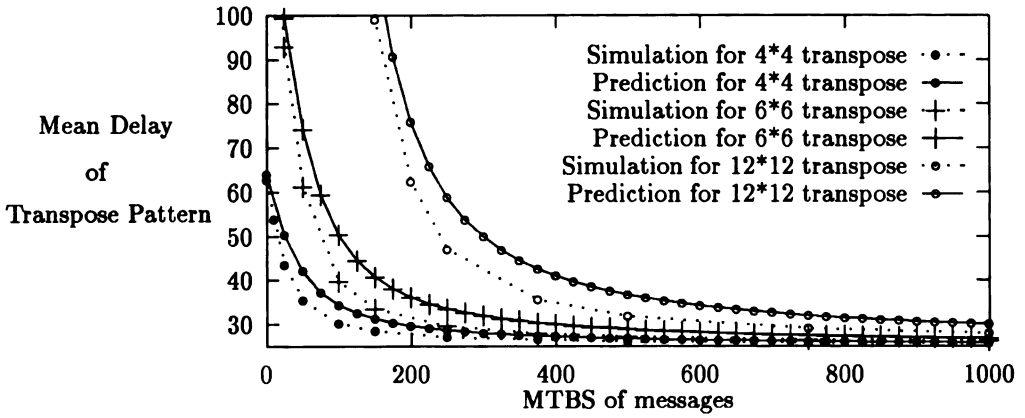


Figure 4.7. Prediction for Matrix Transpose Pattern.

of larger submeshes: 6-by-6 and 12-by-12. The average ICL and SCL of a 6-by-6 transpose pattern are 1.3. For a 12-by-12 transpose pattern the average ICL and SCL are 3.3. Interpolation was used to calculate the contention delays for both patterns. Notice that how well our formulas bound the contention delays found by simulation, and that our formulas provide a very good means for predicting contention for this complex communication pattern.

4.4 Summary

We examined the interactions that occur within a job. We used queueing theory to develop a set of formulas for evaluating internal contention delay of communication paths in a wormhole network. The expected internal contention delay at the jth contention point on ith path was computed by multiplying the probability of a contention at the contention point by the expected contention delay when the contention occurs. The probability of contention with a path can be interpreted as the probability that the channel is used by other competing paths. This probability was estimated by

means of queueing theory. An interesting technique applied in the computation of the contention delay is the separation of the calculation of the expected contention delay at the first starting contention point on a path from the calculation of delay at the subsequent intermediate contention points.

We proposed two metrics that can be used when one wants to measure internal and external contentions between jobs in a multicomputer. These metrics are called the starting contention level and intermediate contention level. Based on the metrics, the internal contention delays of a stair-layered pattern and a complex transpose pattern were predicted and verified by means of simulation. According to the results, as the starting contention level increases, the communication increases for a wide range of communication rates. The amount of increase in communication delay depends on the rate of communication as well as the contention delays facing the external paths that contend at the starting point. Nevertheless, the detrimental delays due to the intermediate contention level primarily occurs only at high rates of communication.

C

A

C

Th
tha
ro
the
lap
inc
tio
wo
m
co
th

co
ar
w
ap

CHAPTER 5

ANALYSIS OF GENERAL CONTENTION DELAY

This chapter provides an analysis of job interactions to predict the contention delay that can occur for a generalized model of job interactions in a 2-D mesh wormhole-routed multicomputer system. The general job interaction model, which is called the *heterogeneous multipath contention model*, is a representation of arbitrarily overlapped communication paths of jobs that have different message injection rates and individual communication patterns. Based on this model, we analyze the degradation of communication performance due to multiple interacting jobs in a 2-D mesh wormhole-routed multicomputer system. We compute the contention delay seen by a message on a path in the heterogeneous multipath contention model. A divide-and-conquer strategy divides the problem into several manageable problems of computing the contention delay for the heterogeneous 2-path contention model.

The rest of this chapter is organized as follows. The heterogeneous multipath contention model and our strategy to analyze the model are presented in Section 5.1 and Section 5.2. Section 5.3 analyzes the heterogeneous two-path contention model, which is used for the analysis of the heterogeneous multipath contention model. Our approach and expressions for predicting the general contention delay due to job inter-

action are presented in Section 5.4. Section 5.5 compares the results of our analysis with simulation. Concluding remarks are given in Section 5.6.

5.1 The Heterogeneous Multipath Contention Model

Our analysis uses a contention model called the *heterogeneous multipath contention model* to study performance degradation due to interactions of several independent jobs. The contention model is a representation of arbitrarily overlapped communication paths of different jobs that have different MTBSs. Suppose that the model has m competing paths that have independent MTBSs, where $m \geq 2$. Let P_1,...,P_m be the m independent competing paths and $mtbs_1$,...,$mtbs_m$ be their MTBSs, respectively. Without loss of generality, the paths in the contention model are assumed to be sorted in the order of the occurrence of the starting contention points, as done in Section 3.3. Figure 5.1 is an illustration of a heterogeneous m-path competing model for our analysis.

Our goal is to predict D^{P_j} for all j, which is the communication delay seen by a message sent through P_j. In our model, D^{P_j} is composed of contention delay and transmission time. The contention delay of D^{P_j}, which is C^{P_j}, can be computed by accumulating $C_i^{P_j}$s for all i, which are the contention delays occurred at the ith contention point on P_j. Note that with the exception of P_m, P_j has j contention points. Therefore, D^{P_j} can be written as follows,

$$D^{P_j} = L * T_t + \sum_{i=1}^{i=j} C_i^{P_j}$$

where T_t is the flit transmission time, L is the number of flits in a message. Since the message transmission time is a known constant when the size of the system is known,

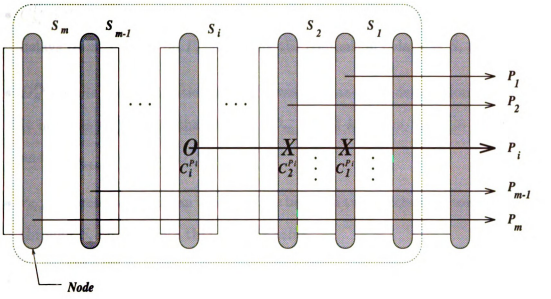


Figure 5.1. The heterogeneous m-path contention model.

what we have to determine are $C_i^{P_j}$, for all i. The rest of this chapter develops the formula to compute $C_i^{P_j}$ and D^{P_j}.

Before proceeding further with the analysis of the heterogeneous m-path contention model, we define the following notation for convenience.

- $P_1,\ldots,P_m$ are the m independent competing paths in the system, whose MTBSs are $mtbs_1$, $mtbs_2$, ..., $mtbs_m$, respectively. As illustrated in Figure 5.1, P_j is assumed to be located at a higher layer than P_k, where $k \geq j+1$.

- S_i is the entrance channel of P_i and is called stage i. The stages are numbered from 1 to m in the reverse direction that the message is sent. S_{i-1} is the stage of the next possible contention after S_i. Between S_i and S_{i-1}, there may be channels that have no contention points. Note that a model with m-paths has m stages.

- D_{tr} is the transmission delay for a message in the model. It is a constant $(= L * T_t)$ in our model.

- $C_{S_i}^{P_k}$ is the contention delay at S_i on P_k. $C_{S_i,S_j}^{P_k}$ is the summation of the contention delays from S_i to S_j on P_k, including the contention delays at S_i and S_j.

- $D_{S_i,d}^{P_k}$ is the communication delay between S_i and the destination node on P_k. It includes $C_{S_i}^{P_k}$. Note that $D_{S_0,d}^{P_k}$ is D_{tr} for all k. Thus, $D_{S_i,d}^{P_k} = C_{S_i,S_1}^{P_k} + D_{tr}$, and $C_{S_i,S_{j+1}}^{P_k} = D_{S_i,d}^{P_k} - D_{S_j,d}^{P_k}$.

- τ_{S_i,P_j} represents the mean waiting time between two successive messages passing through S_i on P_j. Note that τ_{S_i,P_i} is $mtbs_i$.

- λ_{S_i,P_j} is the mean message arrival rates on P_j at S_i, where $1 \leq i \leq j \leq m$.

5.2 The Divide-And-Conquer Strategy

This section presents a divide-and-conquer strategy for predicting the communication delay seen by a message on P_j, a path in the heterogeneous m-path contention model. Consider P_j of Figure 5.1. The number of competing paths of P_j at its starting contention point can be larger than one, but the number of competing paths of P_j at each subsequent intermediate contention point is just one. This is the reason that we compute the starting contention delay separately from the subsequent contention delays. If the several starting competing paths of P_j (i.e., $P_{j+1}, \ldots, P_m$) can be reduced to a single competing path, called Q, that produces a similar amount of contention delay as the original competing paths, then the contention delay seen by a message on P_j can be computed, as in Figure 5.2, by adding the contention delays at the X-marked contention points of the reduced model.Consequently, the problem of computing the contention delay seen by a message on a path in the heterogeneous multipath contention model can be solved by dividing the problem into several smaller problems, each is the computation of the contention delay in a heterogeneous 2-path contention model. To obtain the contention delay on P_j, the contention delays of the

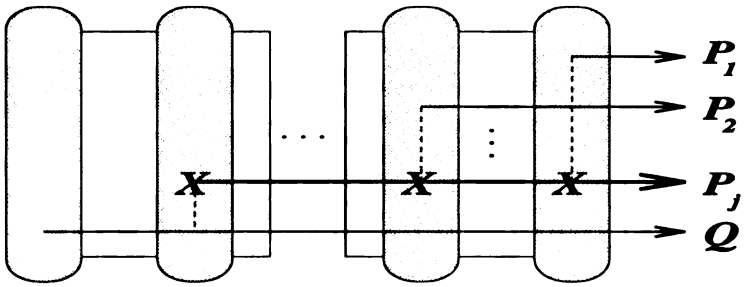


Figure 5.2. The reduced contention model for P_i.

smaller problems are simply added.

The computation of starting or intermediate contention delay in a heterogeneous 2-path contention model is not trivial if the 2-path model is the part of a heterogeneous m-path contention model; the computation for the 2-path contention model requires several parameter values that could be obtained when computing the m-path contention model. This problem is resolved iteratively as follows. Based on the parameter values obtained at the $(n-1)$th iteration, for all P_j the nth iteration computes $D^{P_j}(n)_{S_i,d}$, and $C(n)^{P_j}_{S_i}$ stage by stage from S_1 to S_m in the backward direction. These values are again used for the $(n+1)$th iteration. The iterative computation proceeds until steady state values are reached for $C(n)^{P_j}_{S_i}$. When the iteration reaches steady state, the communication delay seen by a message on P_j is obtained by adding the contention delays at all stages along the path. Computation in the forward direction is impossible because the contention delay at S_i is effected by contentions that occur at S_k for all k greater than i. To construct the basis of the first iteration, the initial iteration computes $D^{P_j}(0)_{S_i,d}$, and $C(0)^{P_j}_{S_i}$ stage by stage, by ignoring all starting contentions but including intermediate contentions.

Section 5.3 presents the detailed analysis and formula to predict the contention delay in a heterogeneous 2-path contention model. Based on Section 5.3, Section 5.4 analyzes the starting and intermediate contention delay on P_j in the heterogeneous

m-path contention model at stage i of the nth iteration, assuming that $D(n-1)^{P_j}_{S_i,d}$ and $C(n-1)^{P_j}_{S_i}$ are known at the $(n-1)$th iteration, for all $1 \leq i \leq j \leq m$.

5.3 Analysis of the Heterogeneous 2-Path Contention Model

This section describes the analysis of the contention delay in a heterogeneous 2-path model as a stand alone model, and not as a part of a heterogeneous m-path contention model. The method in which the derived formula is applied to a heterogeneous m-path contention model is discussed. As explained previously, the analysis in this section is a building block for analyzing contention delays in a heterogeneous m-path contention model. To verify our analysis of the heterogeneous 2-path contention model, we compared our analytical results with a simulation model for several configurations of contention.

5.3.1 Analysis

Suppose that a channel has two competing paths as illustrated in Figure 5.3. Let us name the upper path P_i and the lower path P_j. Both paths have different MTBSs, $mtbs_i$ and $mtbs_j$. We assume that there is only one contention point between the two paths, and it is the first contention point for both paths. The channel where the contention occurs is called H. After the contention point, both paths have their own remaining communication delays, rd_i and rd_j, which are known constants.

Let T_i and T_j be the random variables that represent the local computation times between successive messages at the source nodes of both paths, and let $Cex^{P_k}_{mtbs_i,mtbs_j}$, or simply Cex_k, be the random variable of the external contention delay on P_k (k is either i or j) while MTBSs of P_i and P_j are $mtbs_i$ and $mtbs_j$. Then, the real

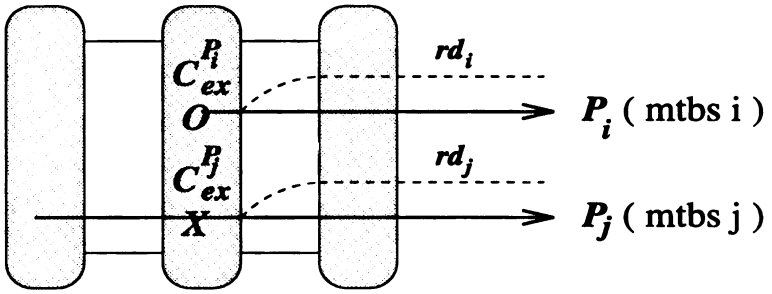


Figure 5.3. Heterogeneous 2-path contention model.

communication delay on P_k is $Cex_k + rd_k$. If $mtbs_i$ is equal to $mtbs_j$, then the 2-path contention model is reduced to a two-layer *stair-layered communication model*, which was proposed in [102]. Thus, we can employ a queuing model as an approximate analytic model to predict the external contention delays in the worst case. However, the queuing approach does not consider the difference between MTBSs of the independent paths, and thus provides an inaccurate estimate of the external contention delay. This section presents a stochastic approach for developing a prediction formula of the external contention delay on P_k.

The expectation of Cex_k can be computed by summing the conditional expectations of Cex_k for all possible conditions that could be made by the relationship between (T_i and T_j) and (rd_i and rd_j). All the possible cases are the following:

Case 1: $T_i < rd_j$ and $T_j < rd_i$ Case 2: $T_i < rd_j$ and $T_j \geq rd_i$

Case 3: $T_i \geq rd_j$ and $T_j < rd_i$ Case 4: $T_i \geq rd_j$ and $T_j \geq rd_i$

and the expectation of the external contention delay on P_k is

$$E[Cex_k] = \sum_{n=1}^{4} Pr\{case_n\} * E[Cex_k \mid case_n] \tag{5.1}$$

where k is either i or j.

The following subsections discuss how to compute the probability and the conditional expectation for each case.

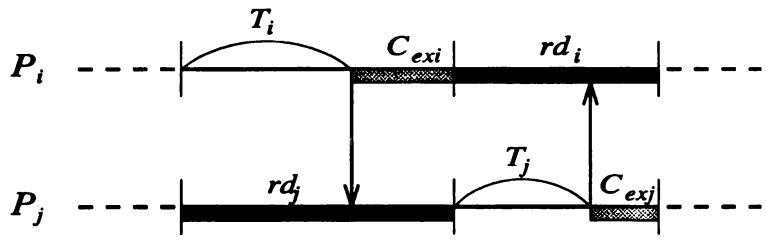


Figure 5.4. An illustration of Case 1.

Case 1

The probability of case 1 is computed as follows:

$$\begin{aligned} Pr\{case_1\} &= Pr\{T_i < rd_j \text{ and } T_j < rd_i\} \\ &= Pr\{T_i < rd_j\}\, Pr\{T_j < rd_i\}, \quad \text{since } T_i \text{ and } T_j \text{ are independent} \end{aligned} \tag{5.2}$$

To compute the conditional expectation of the external contention delay on P_i due to P_j under case 1, it is worthwhile to examine an instance of the case as illustrated in Figure 5.4. Since T_i and T_j are less than rd_j and rd_i respectively, this instance will repeat for the entire overlapping period between P_i and P_j even if T_i and T_j are random variables. Therefore,

$$E[Cex_i | case_1] = rd_j - E[T_i \,|\, T_i < rd_j] \tag{5.3}$$

The reason that $E[T_i \,|\, T_i < rd_j]$ is used instead of $E[T_i]$ is that $T_i < rd_j$ is the given condition in case 1. The computation of $E[T_i \,|\, T_i < rd_j]$ is in the appendix when T_i has an exponential distribution with mean $mtbs_i$ as an example. $E[Cex_j \,|\, case_1]$ can be computed in the same way.

P_i

P_j

Case

The p

rd_{ij} P_i

delay

messa

on P_j

the m

transy

delay

conte

time

the re

an ap

Su

The

the p

that

arriva

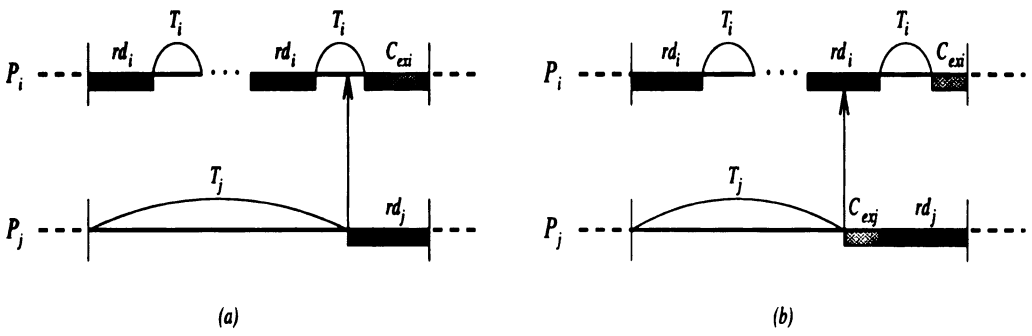


Figure 5.5. An illustration of Case 2.

Case 2

The probability of case 2 is similarly computed as in case 1, i.e., $Pr\{case_2\} = Pr\{T_i < rd_j\}\, Pr\{T_j \geq rd_i\}$. To compute the conditional expectation of the external contention delay under case 2 is complicated. We should consider two possible timings of P_j's message arrival which depends on the status of P_i when the message arrives at H on P_j. One possible situation, as illustrated in the part (a) of Figure 5.5, is that the message of P_j arrives at H while P_i is free. In this case, the message can be transmitted without any extra waiting time, but it causes an external contention delay to a message that arrives at H on P_i during the transmission. The external contention delay, which is named $C_{i_{(a)}}$, is rd_j minus the remaining time of T_i after the time that the message arrives at H on P_j. Our task in computing $C_{i_{(a)}}$ is to calculate the remaining time. By means of renewal theory of stochastic processes, we develop an approximation.

Suppose a process whose mean of inter-arrival time is μ and whose variance is σ^2. The expected remaining time from a given time t until the next event generated by the process is $E[R_t] = \frac{\mu}{2}(1 + \frac{\mu^2}{\sigma^2})$. A proof of this equation is in the appendix. Note that the remaining time is approximately greater than one-half of the mean inter-arrival time. Therefore, the approximate remaining time of T_i can be computed as

$\frac{E[T_i \mid T_i < rd_j]}{2}$, and thus $C_{i_{(a)}}$ is $rd_j - \frac{E[T_i \mid T_i < rd_j]}{2}$. Since T_j is greater than or equal to rd_i, $C_{i_{(a)}}$ is a contention delay after several transmissions. Thus, the expected external contention delay per a message, $E[Cex_i]_{(a)}$, is $\frac{C_{i_{(a)}}}{E[N_t]}$, where $E[N_t]$ is the mean number of transmissions. Approximately,

$$E[N_t] = \lceil \frac{rd_j + E[T_j \mid T_j \geq rd_i]}{rd_i + E[T_i \mid T_i < rd_j]} \rceil \tag{5.4}$$

where N_t is the random variable of the number of transmissions, and $\lceil \rceil$ is the ceiling function that gives the nearest larger integer. In summary, the expected external contention delays per a message on P_i and P_j in the case of part (a) are given in the following equations:

$$E[Cex_i]_{(a)} = \frac{C_{i_{(a)}}}{E[N_t]}, \quad E[Cex_j]_{(a)} = 0 \tag{5.5}$$

The other possible timings of P_j's message arrival is that the message of P_j arrives at H while the channel is used for a message of P_i. This situation is illustrated in part (b) of Figure 5.5. In this case, both paths have contention delays. $C_{i_{(b)}}$ and $C_{j_{(b)}}$ represent the contention delays of P_i and P_j, respectively. $C_{i_{(b)}}$ can be computed in the similar way as in Section 5.3.1, i.e., $C_{i_{(b)}} = rd_j - E[T_i \mid T_i < rd_j]$. As in part (a), $C_{i_{(b)}}$ is a contention delay after several transmissions. The expected external contention delay per message on P_i is $C_{i_{(b)}}$ over the mean number of transmissions, $E[N_t]$. That is,

$$E[Cex_i]_{(b)} = \frac{rd_j - E[T_i \mid T_i < rd_j]}{E[N_t]} \tag{5.6}$$

The external contention delay of P_j, $C_{j_{(b)}}$, is the remaining communication time of P_j after the arrival of the next message. As an approximation, it is one-half of rd_i. Since $C_{j_{(b)}}$ may occur in every transmission, $E[Cex_j]_{(b)} = \frac{rd_i}{2}$.

Let the probability of part (a) be $Pr_{(a)}$, and let the probability of part (b) be $Pr_{(b)}$. The expected external contention delay of P_i and P_j in the case that $T_i < rd_j$

and $T_j \geq rd_i$ are computed as follows:

$$E[Cex_i|case_2] = Pr_{(a)}E[Cex_i]_{(a)} + Pr_{(b)}E[Cex_i]_{(b)}$$
$$E[Cex_j|case_2] = Pr_{(a)}E[Cex_j]_{(a)} + Pr_{(b)}E[Cex_j]_{(b)} \quad (5.7)$$

$Pr_{(a)}$ is the percentage that H is reserved by P_j without any contention delay for several transmissions of P_i. This probability can be approximated by means of renewal theory as follows:

$$Pr_{(a)} = \frac{E[T_i \mid T_i < rd_j]}{rd_i + E[T_i \mid T_i < rd_j]} \quad (5.8)$$

and since $Pr_{(b)}$ is $1 - Pr_{(a)}$, then

$$Pr_{(b)} = 1 - Pr_{(a)} = \frac{rd_i}{rd_i + E[T_i \mid T_i < rd_j]} \quad (5.9)$$

The proof of the equation for $Pr_{(a)}$ is an application of the key renewal theorem. If the channel is considered to be a system that alternates between two states, then a proof is given in [103].

Case 3

Case 3 is exactly the same as case 2 if all notations related to P_i are changed for P_j and all notations related to P_j are changed for P_i. We omit figures and formulas corresponding to this case.

Case 4

The probability of case 4 is $Pr\{T_i \geq rd_j\}\, Pr\{T_j \geq rd_i\}$. Because T_i and T_j are greater than or equal to rd_j and rd_i, respectively, both $E[Cex_i \mid case_4]$ and $E[Cex_j \mid case_4]$ is

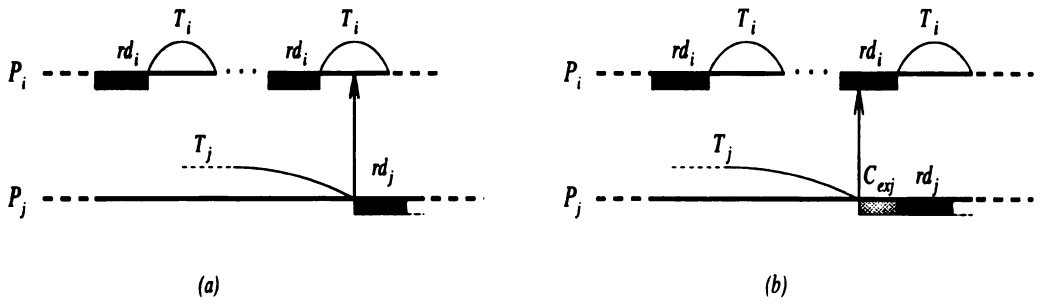


Figure 5.6. An illustration of Case 4.

computed by using renewal theory as we do in Case 2. Figure 5.6 illustrates how to compute $E[Cex_i \mid case_4]$. As before, we consider two possible situations depending on the relative timing of the messages of both paths. If a message of P_j arrives during T_i as in the part (a) of Figure 5.6, then the waiting time of the message is zero. If a message of P_j arrives during rd_i as in part (b), then the waiting time is approximately one-half of rd_i. According to the renewal theory, the probability of the part (a), $Pr_{(a)}$, is $\frac{E[T_j \mid T_j \geq rd_i]}{rd_j + E[T_j \mid T_j \geq rd_i]}$, and the probability of the part (b), $Pr_{(b)}$, is $1 - Pr_{(a)}$. Therefore, the expected external contention delay of P_i is

$$E[Cex_i \mid case_4] = Pr_{(b)} * \frac{rd_i}{2} \tag{5.10}$$

Similarly, $E[Cex_j \mid case_4]$ can be computed

$$E[Cex_j \mid case_4] = \frac{rd_i}{rd_i + E[T_i \mid T_i \geq rd_j]} * \frac{rd_j}{2} \tag{5.11}$$

In summary, the external contention delay due to the interaction between P_i and P_j can be computed by Equation 5.1 with the input values, $mtbs_i$, $mtbs_j$, rd_i, and rd_j when the two paths are not a part of a larger m-path contention model. However, if there are other competing paths in the multipath contention model where the two

paths are involved, input values should be the values that include the delays due to the contentions with the other paths. Thus, τ_{S_k,P_i}, τ_{S_k,P_j}, $D^{P_i}_{S_{k-1},d}$, and $D^{P_j}_{S_{k-1},d}$, which are defined in Section 5.1, are used instead of $mtbs_i$, $mtbs_j$, rd_i, and rd_j respectively, where S_k indicates the stage at which the 2-path contention situation is considered. Based on that equation, in the next section we analyze the heterogeneous multipath contention model. Equation 5.1 is used as a known function called $f()$. This function returns $E[Cex^{P_i}_{\tau_{S_k,P_i},\tau_{S_k,P_j}}] + D^{P_i}_{S_{k-1},d}$ as $f_i()$ and $E[Cex^{P_j}_{\tau_{S_k,P_i},\tau_{S_k,P_j}}] + D^{P_j}_{S_{k-1},d}$ as $f_j()$.

5.3.2 Results

We present results that demonstrate how well our analysis of the heterogeneous 2-path contention model compares with a simulation model for several configurations of contention. We used the same Multisim [98] simulator as before.

The simulation model is composed of two competing paths, P_i and P_j, which are parts of two independent homogeneous multipath contention models. The homogeneous multipath contention models for P_i and P_j have different MTBSs, $MTBS_i$ and $MTBS_j$, and m and n competing paths respectively. Figure 5.7 illustrates the simulation model. We call the simulation model the $m-n$ interaction model. For simulation purposes, the remaining communication times of P_i and P_j after the external contention are assumed to be their own internal contention delays that are independent of the other path. Based on the analysis in Chapter 4, the internal delays are assumed to be a known value for the given MTBS and the given communication pattern. The remainder of this subsection shows the effects of interactions of P_j on the performance of the P_i as the MTBSs and the numbers of the competing paths of the homogeneous models are various. Since there is only one external contention point between P_i and P_j in this simulation model, the effects of P_i on the performance of P_j are identical.

Figure 5.8 shows the mean communication delay of P_i as a function of the $MTBS$ of P_i as different values of the $MTBS$ of P_j are used. The $1-1$ contention model

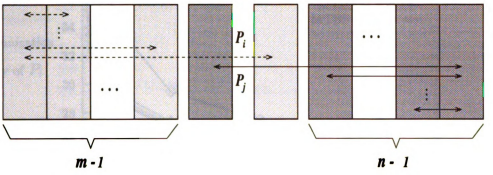


Figure 5.7. The m-n simulation model for 2-path contention.

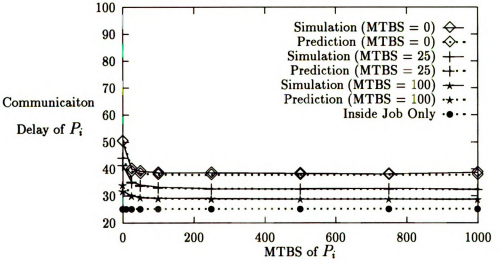


Figure 5.8. Performance effect of various MTBSs of P_j on communication of P_i: Mean communication delay of P_i vs.MTBS of P_i for 1-1 interaction model while MTBS of P_j is 0, 25 or 100.

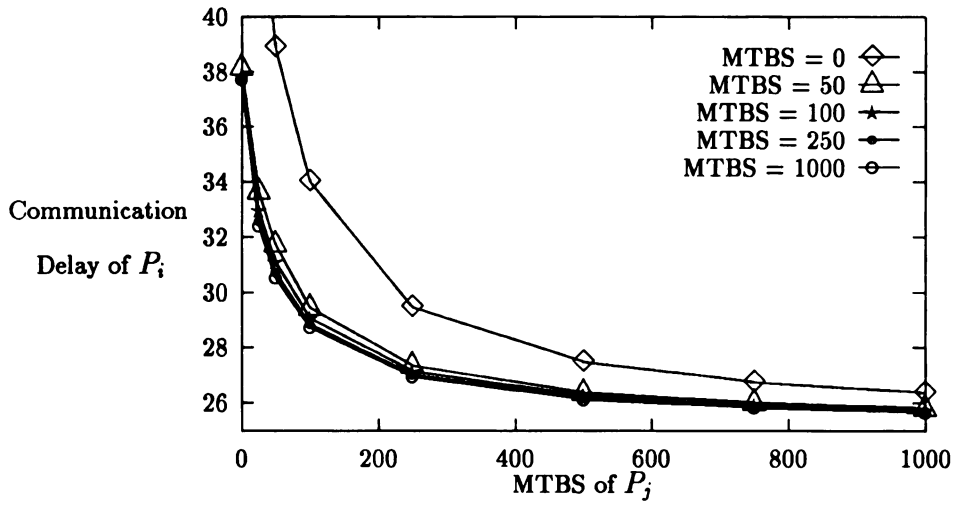


Figure 5.9. Performance effect of various MTBSs of P_j on communication of P_i : Mean communication delay of P_i vs.MTBS of P_j for 1-1 interaction model while MTBS of P_i is 0, 25, 100, 250 or 1000.

is employed for the simulation. Since P_i does not have any internal contention, the communication delay of P_i without interactions with P_j is merely the transmission delay, i.e., 25 μsec, as shown on the lowest plot of Figure 5.8. The only cause for an increase in $P_i's$ communication delay is its contentions with P_j. The other plots in Figure 5.8 show the effect of P_j on the performance of P_i. As the communication rate of P_j increases, the communication delay of P_i increases for all ranges of communication rates of P_i. It increases more significantly for high rates. This relationship between $MTBS$ of P_j and the mean communication delay of P_i can be seen more clearly in Figure 5.9. As the $MTBS$ of P_j becomes smaller (i.e., the communication rate becomes larger), the analytical results show that the mean communication delay of P_i becomes larger.

Another factor that has an effect on the communication time of P_i is the number of competing paths of the homogeneous contention model in which P_j resides. As we have examined in Chapter 4, the internal contention delay of the path increases as the

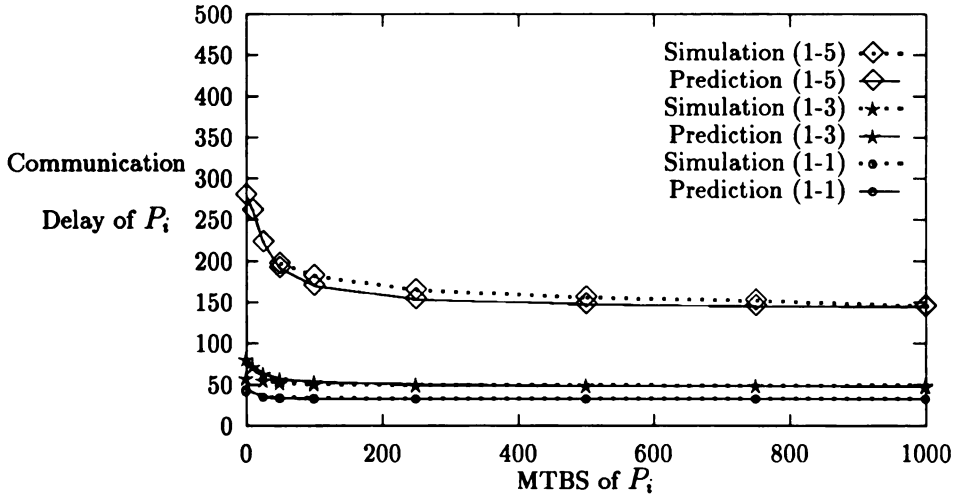


Figure 5.10. Performance effect of number of layers of P_j on communication of P_i : Mean communication delay of P_i vs.MTBS of P_i for 1-1, 1-3 and 1-5 interaction model : $MTBS_2 = 25$.

number of layers becomes large. This increase in the internal contention delay can effect the performance of P_i. Figures 5.10 and 5.11 present the effect of the increase in the number of layers in which P_j resides on the performance of P_i. Figure 5.10 shows the mean communication delay of P_i which is measured as a function of the $MTBS$ of P_i, while P_i has a path that shares a competing channel with P_j that resides with 1, 3 or 5 layers. To provide insight to their relationship, the mean communication delay of P_i is given in Figure 5.11. The delay is shown as a function of the number of layers associated with P_j for several combinations of $MTBS_1$ and $MTBS_2$. The number of layers associated with P_j is varied from 0 to 7, while that of P_i is fixed to 1. As we see in the figure, the communication of P_i is not affected by the communication of P_j, when P_j sends message slowly at around 500 μsec. Nevertheless, at high rates such as 25 μsec, the communication delay of P_i increases greatly for all ranges of MTBS of P_i. Figure 5.12 is an another view that shows the relationship among MTBSs of P_i and P_j and the number of layers. At extremely small values of MTBS of P_i, a small decrease in the MTBS of P_j or a small increase in the number of layers causes very

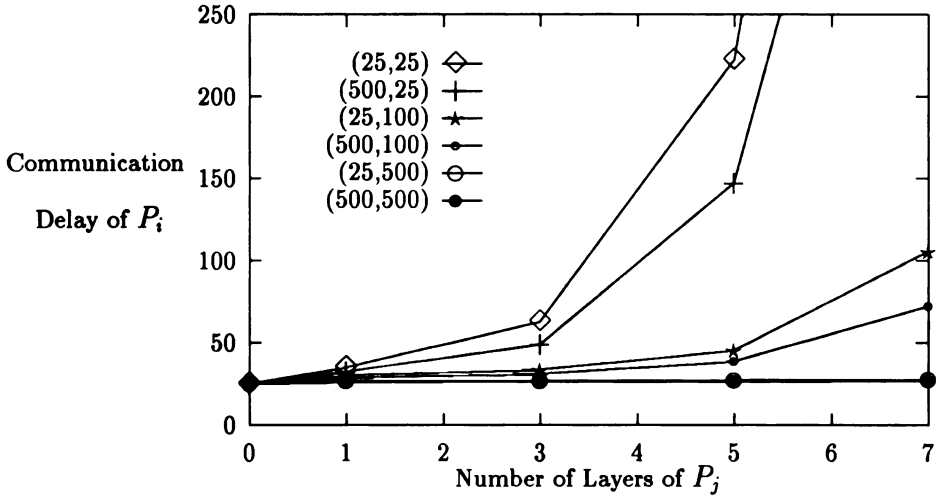


Figure 5.11. Performance effect of number of layers of P_j on communication of P_i : Mean communication delay of P_i vs.number of layers of P_j. The pair represents the MTBSs of P_i and P_j.

significant interactions. At other rates of MTBS of P_i, the decrease of MTBS of P_j or an increase in the number of layers of P_j increases the communication time of P_i less significantly.

5.4 Analysis of the Heterogeneous Multipath Contention Model

This section completes the divide-and-conquer strategy in Section 5.2 for predicting the real communication delay seen by a message on a path in the heterogeneous m-path contention model. Based on the analysis of the 2-path contention model, this section derives formulas which compute the contention delay of $P_j (i \leq j \leq m)$ at stage i in the nth iteration.

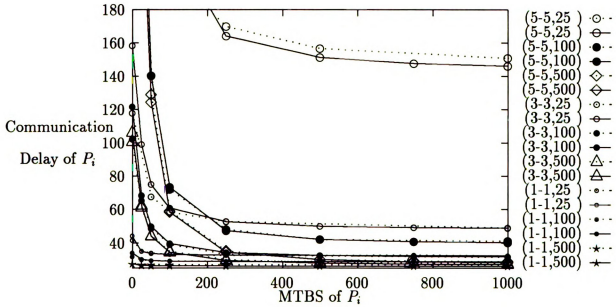


Figure 5.12. Performance effect of combination of factors on communication of P_i: dotted line-simulation results, solid line-predictions. The first element of the pair represents k-k interaction model and the second number represents the MTBS of P_j.

5.4.1 Starting Contention Delay

Figure 5.13 illustrates the situation involving starting contention for a message on P_i at stage i. As mentioned in Section 5.2, $\tau(n-1)_{S_i,P_j}$ and $D(n-1)^{P_j}_{S_i,d}$ are assumed to be already computed at the $(n-1)$th iteration, for all i, j such that $1 \leq i \leq j \leq m$. The number of the starting competing paths of P_i (i.e., $P_{i+1}, \ldots, P_m$) is $m-i$, as shown in part (a) of Figure 5.13. This number may be larger than one. Recall that in order to apply $f()$ there should be only two competing paths at the contention point, whose τ and the remaining communication delays after the contention are known at the considered stage. Therefore, our strategy for computing the starting contention delay of P_i at S_i in the nth iteration takes the following steps:

1. Identify τ of P_j at S_i in the nth iteration, i.e., $\tau(n)_{S_i,P_j}$, for all j such that $i \leq j \leq m$.

2. Reduce the multiple starting competing paths of P_i to a single competing path (say Q) that generates the same amount of starting contention delay.

3. Identify the remaining communication delays after the contention for both P_i and Q in the nth iteration, i.e., $D(n)^{P_i}_{S_{i-1},d}$. and $D(n)^{Q}_{S_{i-1},d}$.

4. Apply $f_i()$ to compute $D(n)^{P_i}_{S_i,d}$.

In the first step, $\tau(n)_{S_i,P_i}$ is simply $mtbs_i$. Let P_j be one of the starting competing paths of P_i, where $i+1 \leq j \leq m$. Due to the contention points that are on P_j before stage i,

$$\tau(n)_{S_i,P_j} = mtbs_j + C(n-1)^{P_j}_{S_j,S_i}$$

where $i+1 \leq j \leq m$.

For the second step, we use an approximation as follows. Let A_{S_i,P_j}, $i+1 \leq j \leq m$, be the random variable of the message arrivals of P_j at stage i, whose mean is

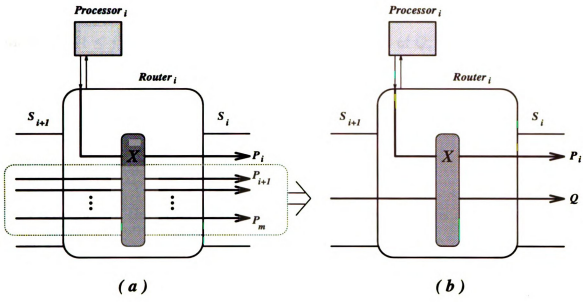


Figure 5.13. Computation of starting contention delay.

λ_{S_i,P_j}. If A_{S_i,P_j} follows a Poisson distribution, then the mean message arrival rate generated at the starting channel of P_i by the $m-i$ starting competing paths of P_i, is $\sum_{l=i+1}^{m} \lambda_{S_i,P_l}$ [104]. In fact, we do not know the actual distribution of A_{S_i,P_j}, but the simulation results in the Section 5.4.3 verify that this approximation serves us as a good estimate. Therefore, we substitute a competing path, say Q, for the $m-i$ starting competing paths of P_i, as in part (b) of Figure 5.13, and

$$\tau(n)_{S_i,Q} = 1/\sum_{j=i+1}^{m} \lambda_{S_i,P_j} - D(n-1)^{Q}_{S_i,d}$$

where $D(n-1)^{Q}_{S_i,d}$ is the value defined in the next step of the $(n-1)$th iteration. In our model, $\lambda(n)_{S_i,P_j}$ is computed as $1/(\tau(n-1)_{S_i,P_j} + D(n-1)^{P_j}_{S_i,d})$.

The last step identifies the remaining communication delays on P_i and Q after the contention at stage i. For the remaining communication delays on P_i in the nth iteration, $D(n-1)^{P_i}_{S_{i-1},d}$ can be used. The remaining communication delay on Q, which is $D(n)^{Q}_{S_{i-1},d}$, is approximated as a weighted average of the remaining commu-

nication delays on $P_{i+1}, \ldots, P_m$ after the contention at stage i. A reasonable weight for P_j, $i+1 \leq j \leq m$, is P_j's contribution to the arrival of Q, i.e., ratio $\lambda(n)_{S_i,P_j}$ to $\sum_{l=i+1}^{m} \lambda(n)_{S_i,P_l}$. Thus,

$$D(n)^{Q}_{S_{i-1},d} = \sum_{j=i+1}^{m} \frac{\lambda(n)_{S_i,P_j}}{\sum_{l=i+1}^{m} \lambda(n)_{S_i,P_l}} * D(n-1)^{P_j}_{S_{i-1},d}$$

Consequently, the expected real communication delay seen by a message on P_i at stage i is,

$$D(n)^{P_i}_{S_i,d} = f_i(\tau(n)_{S_i,P_i}, \tau(n)_{S_i,Q}, D(n)^{P_i}_{S_{i-1},d}, D(n)^{Q}_{S_{i-1},d})$$

and the expected starting contention delay of P_i is

$$C(n)^{P_i}_{S_i} = D(n)^{P_i}_{S_i,d} - D(n)^{P_i}_{S_{i-1},d}$$

5.4.2 Intermediate Contention Delay

Consider a situation of an intermediate contention on P_j at stage i, where $i+1 \leq j \leq m$. Figure 5.14 illustrates the case when j is $i+1$. P_i is the only competing path of P_{i+1} at stage i. Therefore, we can apply $f()$ without the second step of reducing the number of competing paths. Again, $\tau(n-1)_{S_i,P_j}$ and $D(n-1)^{P_j}_{S_i,d}$ are known values at the $(n-1)$th iteration for all i, j such that $1 \leq i \leq j \leq m$.

The first step is the same as for the starting contention delay. That is, $\tau(n)_{S_i,P_i}$ is simply $mtbs_i$, and $\tau(n)_{S_i,P_j} = mtbs_j + C(n-1)^{P_j}_{S_j,S_i}$, where $i+1 \leq j \leq m$.

The last step for identifying the remaining communication delays on P_i and P_j after the contention at stage i is more difficult than for the starting contention delay. Without loss of generality, suppose that j is $i+1$ as in the part (b) in Figure 5.14. For the remaining communication delays on P_{i+1} after S_i, $D(n-1)^{P_{i+1}}_{S_{i-1},d}$ can be used. However, the remaining communication delays for P_i may be larger than $D(n-1)^{P_i}_{S_{i-1},d}$,

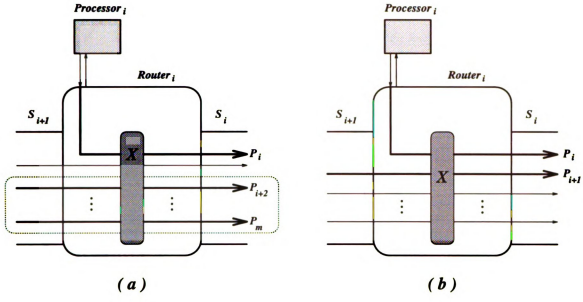


Figure 5.14. Computation of intermediate contention delay.

since the other competing paths at stage i, i.e., $\{P_{i+2}, \ldots, P_m\}$, may increase the delay. Thus, the contention delay between P_i and P_{i+1} with $\{P_{i+2}, \ldots, P_m\}$ must be larger than the delay without $\{P_{i+2}, \ldots, P_m\}$. The difference can be interpreted as the possibility of contention between P_i and $\{P_{i+2}, \ldots, P_m\}$. Therefore, the remaining communication delays on P_i increases as much as the contention delay between P_i and $\{P_{i+2}, \ldots, P_m\}$. The contention delay can be computed in the same way as the computation of the starting contention delay of P_i, which is described in Section 5.4.1, except that there are $m-i-1$ starting competing paths for P_i, i.e., $P_{i+2}, \ldots P_m$. Thus, the modified remaining communication delays on P_i after stage i for computing the intermediate contention delay on P_j at stage i is

$$\hat{D(n)}^{P_i}_{S_{i-1},d} = f_i(\tau(n)_{S_i,P_i}, \tau(n)_{S_i,Q_{(i+1,m),\neq j}}, D(n)^{P_i}_{S_{i-1},d}, D(n)^{Q_{(i+1,m),\neq j}}_{S_{i-1},d})$$

Therefore, the expected real communication delay on P_j from stage i to stage 1 is

$$D(n)^{P_j}_{S_i,d} = f_j(\tau(n)_{S_i,P_i}, \tau(n)_{S_i,P_j}, \hat{D(n)}^{P_i}_{S_{i-1},d}, D(n)^{P_j}_{S_{i-1},d})$$

and the intermediate contention delay on P_j at stage i is

$$C(n)^{P_j}_{S_i} = D(n)^{P_j}_{S_i,d} - D(n)^{P_j}_{S_{i-1},d}$$

5.4.3 Results

In this subsection we provide results that show agreement between the analysis and a simulation of the heterogeneous multipath contention model. In order to illustrate the comparisons for the multipath contention model, we show analytical and simulation results for a heterogeneous 5-path contention model.

Figure 5.15 shows the average communication delays of the five paths of a 5-path contention model. We analyzed many different cases, and present in this figure the results for fixed values of MTBS for all paths in the model with the exception of path 3, the middle path in the model. The respective values of the MTBS for each path represented in this figure, from path 1 to path 5, are $0, 50, k, 250, 500$ μsecs, where k varies from 0 to 500 μsecs.

We observe how the communication delay relates to the MTBS. As the MTBS decreases, the mean communication delay increases because the probability of contention increases the average path delay. Path 1 is a path that has no intermediate contention points and has four other competing paths at its starting contention point in the heterogeneous 5-path contention model. In contrast, path 5 has four intermediate contention points and no starting contention point. Notice that the paths that have the greater number of intermediate contention points are more sensitive to the network load than the paths that have fewer intermediate contention points. Path 1

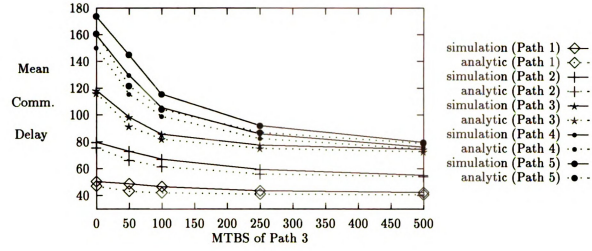


Figure 5.15. Mean communication delay of each path vs. MTBS of Path 3 for heterogeneous 5-path contention model. The MTBS of Paths 1, 2, 4, 5 are 0, 50, 250, and 500 μsecs, respectively.

is relatively unaffected by the network load.

5.5 Analyzing Two Transpose Jobs

We apply our analytical model to a more complex job-interaction model, which is illustrated in Figure 5.16. Two jobs are interleaved and have overlapping communication paths. The first job is allocated to a contiguous partition of processors and is placed inside of the second job. The second job has its processors scattered to two contiguous regions outside of the first job. The overlapping communication paths are illustrated. The logical communication pattern of each job is a 4-by-4 transpose pattern, which has been used by other researchers to study performance effects of processor mappings [101]. In this pattern, a subset of nodes actively send messages. Only the nodes on the diagonal do not send messages. Each active node has a related node with which it exchanges messages repeatedly. The communicating pairs

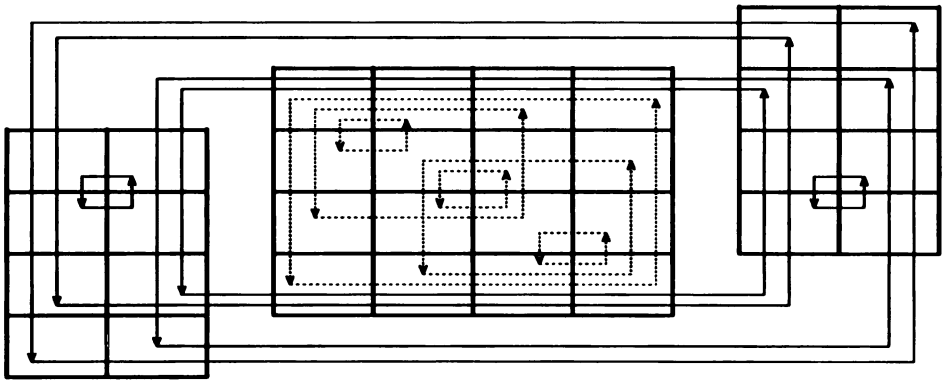


Figure 5.16. An allocation of two transpose jobs.

of nodes using the matrix-transpose pattern are specified as $Node_{i,j} \longleftrightarrow Node_{j,i}$, where $Node_{k,l}$ indicates the node in the submesh at the kth row and lth column. The symbol $\longleftrightarrow$ indicates bi-directional communication. Note that the physical communication pattern of the outside job is different from its logical communication pattern. In fact, the physical communication pattern of the scattered job is determined by the location of the processors allocated to the job.

The job interaction model of this example can be divided into several heterogeneous multipath contention models, row by row. Each row has a pair of heterogeneous multipath contention models; one model is directed to the left and the other model is directed to the right. In other words, there are 6 left-directed heterogeneous multipath contention models and 6 right-directed heterogeneous multipath contention models. To illustrate the effects of communication delays produced by the job interaction model, we compute the average communication delays for each path in each direction separately, and then average the delays. In this example, the communication pattern of each direction is symmetric and therefore will be the same in each direction. We thus compute the average communication delay of each job only for the right direction. Let (i, j) be a heterogeneous $(i + j)$-path contention model constructed by i paths that belong to the inside job and j paths that belong to the outside

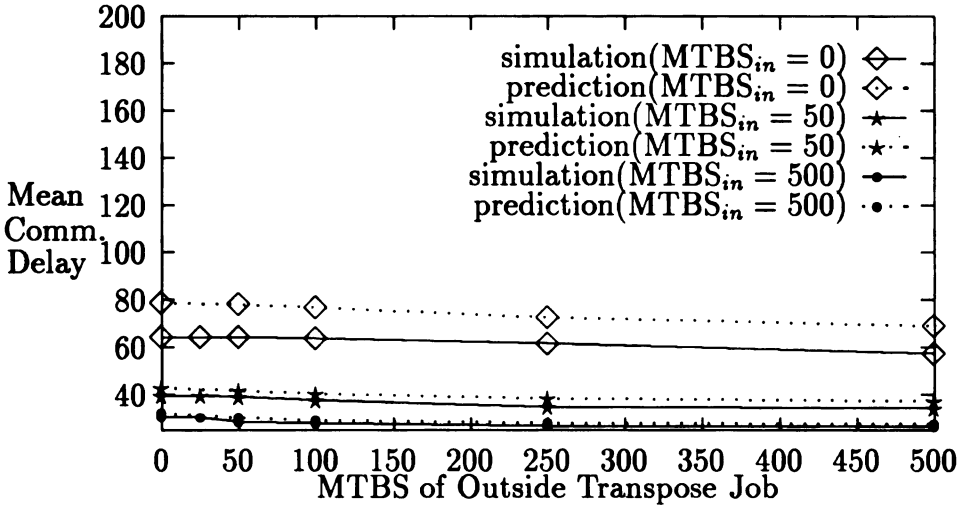


Figure 5.17. Mean communication delay of the inside transpose job while the MTBS of the outside job varies from 0 to 500μsecs. $MTBS_{in}$ is MTBS of the inside job.

job. Using this notation, the 6 right-directed contention models can be represented as (0,0),(0,0),(1,0),(2,0),(3,2), and (0,2) from the top row to the bottom row.

The simulation and analytical results are given in Figures 5.17 and 5.18. Figure 5.17 shows the average communication delay of the inside job for various values of MTBS of the inside job as a function of the load of the outside job. Figure 5.18 shows the delay of the outside job for the same set of parameters. The simulation results are verified with the computation of the analytical model for all cases.

5.6 Summary

We have analyzed the performance degradation due to the sharing of network resources by multiple independent interacting jobs for a general contention model called the *heterogeneous multipath contention model.* Our analysis is based on a divide-and-conquer strategy, which derives the communication delay at each contention point on

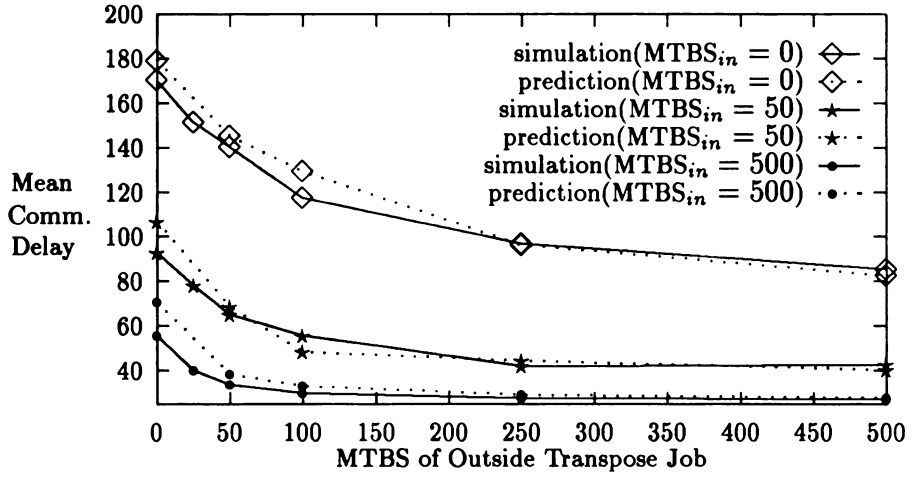


Figure 5.18. Mean communication delay of the outside transpose job while the MTBS of the outside job varies from 0 to 500μsecs. $MTBS_{in}$ is MTBS of the inside job.

a path. It reduces a heterogeneous multipath contention model at each contention point to a heterogeneous 2-path contention model. The computation of the reduced model distinguishes the starting contention point from the intermediate contention points. Simulations indicate that our analysis of the analytic model closely agrees with the results of the simulations. These results help us understand the effect of job interactions on network performance such that we are better prepared for finding solutions to the problem of allocating processors in 2-D mesh wormhole-routed multicomputers.

CH

JO

PI

In thi

proce

impor

enjoy

effect

job in

the in

measu

is a m

wheth

of job

alloca

analy

Th

paths

sidera

CHAPTER 6

JOB PARTITIONING AND PLACEMENT

In this chapter we study principles that may be applied when developing a scattered processor allocation strategy for a 2-D mesh multicomputer. We believe that it is important to develop scattered processor allocation strategies for MPCs in order to enjoy the increased processor utilizations that they allow as long as the negative effects of job interactions can be kept under control. We investigate the effects of job interactions due to communication parameters such as the communication rate, the internal competing level, and the congestion factor. The competing level is a measure of the contention on a path within an individual job. The congestion factor is a measure of the contention at a channel. By isolating each parameter, we study whether the method of partitioning and placing jobs can change the negative effects of job interactions. Our investigation of factors that affect a scattered processor allocation strategy uses a combination of simulations and an analytic model that is analyzed in Chapter 5.

The rest of the chapter is organized as follows. Effects of starting competing paths and intermediate competing paths are examined in Section 6.1 with the consideration of contention among competing paths. In Section 6.2, efficient methods for

part

Con

6.1

In or

comm

comp

how

a pat

worm

the re

simul

the s

Resul

the ch

Su

differe

proce

chann

MTBS

of the

our ini

in Fig

and (b

which

upper

partitioning and placing jobs and effects of communication parameters are studied. Concluding remarks are given in Section 6.3.

6.1 Contention Among Competing Paths

In order to investigate the effects of contention between jobs that have independent communication characteristics, we must first study the effects of contention among competing paths that have independent communication rates. This section examines how contention between a set of competing paths can affect the delays experienced by a path. The insight obtained from this examination is intuitive from the nature of the wormhole routing switching technique, but it is basic and substantial to understand the results of job placement and partitioning in the next section. This section presents simulation results and the explanations of the results. For our simulation, we used the same Multisim [105] simulation package as mentioned in the previous chapters. Results provided by an analytical technique described in Chapter 5 can help explain the characteristic of the results.

Suppose that we have five competing paths, P_1, P_2, P_3, P_4, and P_5, which have different MTBSs, as illustrated in Figure 6.1. In the figure, the gray ovals represent processing nodes and the white rectangles between two consecutive ovals represent the channels between two consecutive processing nodes. P_1, P_2, P_4 and P_5 have constant MTBSs, but the MTBS of P_3 varies from 0 to 500. The overall communication delay of the communicating paths may depend on how the paths overlap. For the purpose of our initial investigation, we consider two possible layouts. The layouts are illustrated in Figure 6.1 by the Mean Time Between Sends (MTBSs) listed in the columns (a) and (b), which we will label as layout (a) and layout (b). Layout (a) is the case in which a path with a higher communication rate (*i.e.*, smaller MTBS) is located at the upper level among the paths, and the paths with the lower communication rates (*i.e.*,

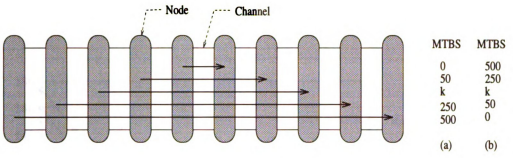


Figure 6.1. Two layouts of 5 competing paths.

larger MTBSs) are located at the lower paths. In contrast, layout (b) places a path with a lower communication rate (*i.e.*, larger MTBS) at a upper level, while the paths with higher communication rates are located at lower levels. As shown in the figure, the set of communication rates used in the study are identical for both layout (a) and layout (b). The only difference is the location of the paths with different MTBSs.

Figure 6.2 shows the average communication delays of the five paths in layout (a) whose MTBSs (in μsec) are $0, 50, k, 250$, and 500, where k varies from 0 to 500. The communication delays are displayed as functions of MTBS of P_3. Figure 6.3 has the results for layout (b), which has the MTBSs for the paths to be $500, 250, k, 50$, and 0. P_3 has an MTBS that varies from 0 to 500. All dotted lines in Figures 6.2 and 6.3 present our results from the analytical model, and the solid lines present our simulation results. As displayed by the figures, the communication delays for all paths of layout (b) are smaller than that of layout (a). This result implies that the overall communication delay can be reduced by positioning the paths in a particular order, such as layout (b). This phenomenon can be explained by understanding the detailed interactions between the competing paths.

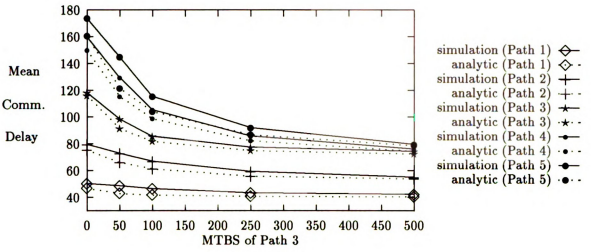


Figure 6.2. Communication delay of layout (a) vs. MTBS of P_3 that varies from 0 to 500: MTBSs of P_1, P_2, P_4, and P_5 are $0, 50, 250$, and 500, respectively. The mean communication delay is greater for layout (a) than the delay for layout (b) shown in the next figure.

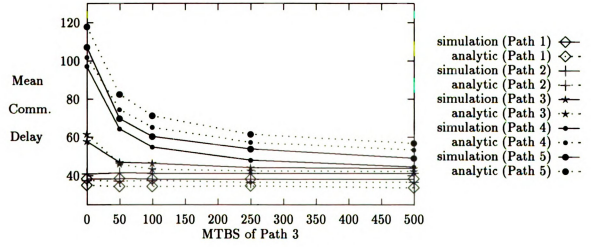


Figure 6.3. Communication delay of layout (b) vs. MTBS of P_3 that varies from 0 to 500: MTBSs of P_1, P_2, P_4, and P_5 are $500, 250, 50$, and 0, respectively. The delay shown in this figure is less than the delay of layout (a) shown in the previous figure.

6.

In
sta
an
pa

EF

Sup
cha
bu
sta
a n
sev
can
occ
con

Th
pet
pat
is
MI
its
pa
as
oth
wa

6.1.1 Path Interaction

In view of analyzing path interactions, this subsection re-interprets the effects of starting contention and intermediate contention by considering the number of starting and intermediate competing paths and the communication rate of the competing paths.

Effect of Starting Competing Paths

Suppose path P starts at the processor that is connected to the router for the specific channel that we are examining, and other paths route messages through the channel (but start at other processors in the network) and will be competing paths to P. A starting contention of P occurs if P sends a message while the channel is utilized by a message from one of the starting competing paths. Even if there are potentially several starting competing paths, only one path can contend at a time since a channel can be used by only one path at a time. The probability that starting contention occurs is greater as the number of the starting competing paths increases and the communication rates of the starting competing paths increase.

Figure 6.4 shows the starting contention delay of a path as a function of its MTBS. The figure shows results in which a path that has contention at its starting path competes with 3 or 5 paths. Results are displayed when the MTBSs of starting competing paths are $0, 50, 100, 250$, and 500. The results show that starting contention delay is relatively insensitive to the change in the number of competing paths and their MTBSs. This can be explained by considering a message that is initially blocked at its starting channel because the starting channel is utilized by one of the competing paths. The message at its starting channel will be able to acquire the channel as soon as the tail flit of the message utilizing the competing path passes the channel. No other competing path will acquire the channel as quickly as a path with a message waiting at its starting channel. Therefore, an increase in the number of competing

Mea
Comm
Dela

Figur
paths

paths

Effec

The s
that a
tentio
shows
diate
In con
very s
media
or an
effect
the c

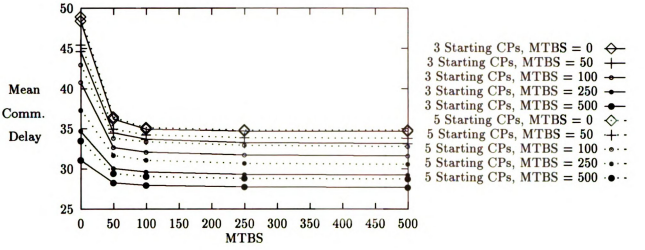


Figure 6.4. The starting contention delay when competing with 3 or 5 competing paths (CPs) at a starting channel.

paths does not have a great effect on a path's access to its starting channel.

Effect of Intermediate Competing Paths

The starting channel of a path is also an intermediate contention point for channels that are routed through the starting channel. A path has as many intermediate contention points as the number of the intermediate competing paths it faces. Figure 6.5 shows the intermediate contention delay of a path that faces either 3 or 5 intermediate contention points. We assume that the path has no starting contention delay. In contrast to the effect of starting contention delay, intermediate contention delay is very sensitive to the increase of the communication rate and the number of the intermediate competing paths. This is because an additional intermediate competing path or an increased communication rate of an intermediate competing path has a chain effect by increasing the delay of all other intermediate competing paths. Therefore, the characteristics of intermediate competing paths affect the delay of a path more

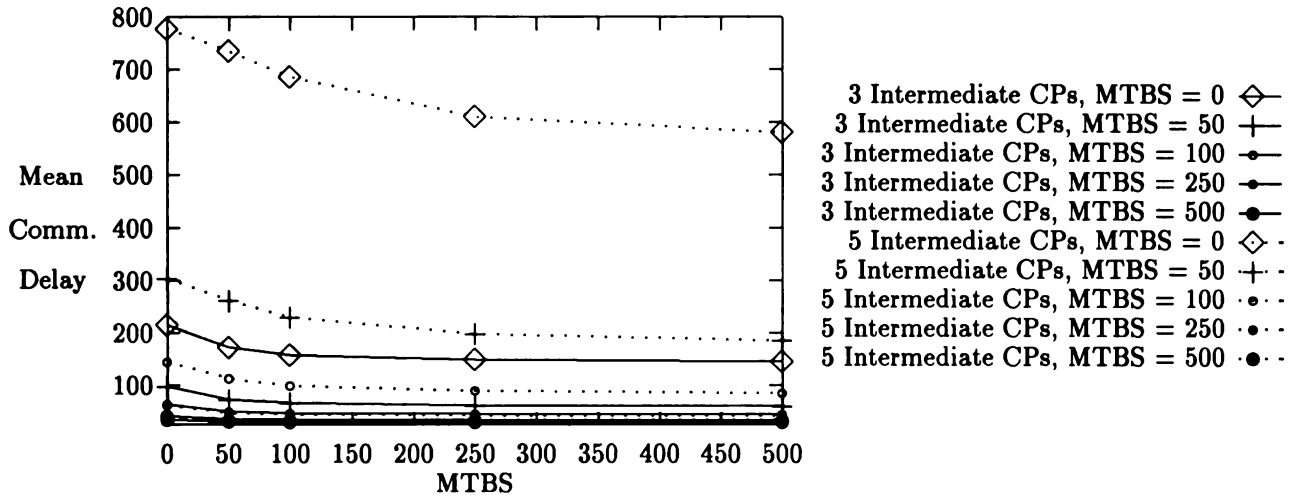


Figure 6.5. The intermediate contention delay when competing with 3 or 5 intermediate competing paths (CPs).

significantly than the characteristics of the starting competing paths.

6.1.2 Discussion

Let us reconsider the paths illustrated in Figure 6.1. The only difference between layouts (a) and (b) is that the communication rates of upper paths and lower paths. The higher communication rates of the lower paths contribute to the increase of the starting contention delays of the upper paths. The starting contention delays, however, are relatively insensitive to the increased communication rate. In contrast, the lower communication rates of the upper paths contribute to the decrease in the intermediate contention delay of the lower paths. The intermediate contention delay is very sensitive to a change in communication rate. Therefore, the overall performance of layout (b) is better than that of layout (a). In general, if you have control of the layout of communicating paths, it is much better to cause the paths with greater communication demands to encounter a greater number intermediate contention points

re
inf
am
pa

6.

Ou
ne
in
ca
ac

6.

We
we
the
pa
ha
the
pa
no
are
0.
pa
be
tha

relative to the number of intermediate contention points encountered by paths that infrequently transmit messages. It is undesirable to have the paths with a greater amount of communication demand to be intermediate contention points for other paths.

6.2 Partitioning and Placing Jobs

Our investigation of the effect of contention among multiple paths in a wormhole network provides a basis for our investigation of the interferences that occur among interleaved jobs. This investigation is important in order to understand how to allocate jobs to a 2-D mesh system when the processors allocated to a job can be scattered across the system.

6.2.1 Methodology

We constrain our study to examine how two interleaved jobs interact. For simplicity we assume that all paths that belong to a job have the same communication rate and the job has a specific pattern in which messages are transmitted. We consider two patterns of communication. A *diagonal pattern* is used when we want a given job to have no internal contentions, such that the only contentions from which a job suffer are the contentions due to other jobs. The communicating pairs of nodes in the diagonal pattern are specified as $Node_{i,i} \longleftrightarrow Node_{(n-i-1),(n-i-1)}$, where $Node_{k,l}$ indicates the node in the submesh at the kth row and lth column. The specification assumes there are n rows and n columns assigned to a job, with rows and columns numbered from $0 \ldots n-1$. The symbol $\longleftrightarrow$ indicates bi-directional communication. The second pattern of communication that we use is the matrix-transpose pattern, which has been introduced in Chapter 4. This pattern is used when we want to consider cases that the internal contention within a given job is significant. The two patterns of

(a) Diagonal (b) Transpose

Figure 6.6. Two logical communication patterns within a 4x4 job.

communication are illustrated in Figure 6.6. These two communication patterns are very simple, but by using these patterns we form job interaction scenarios that can isolate the effect of several communication parameters due to the job interactions. In Section 6.2.5, we discuss how to apply the results that we obtain by the simulation of two interleaved jobs for developing scattered processor allocation strategies.

Suppose that two jobs should be allocated to an MPC. Consider a situation in which we cannot allocate both jobs to contiguous regions of the MPC due to the lack of available processors in partitions that are large enough for both jobs. What will be the effect of one job on the other if one of the jobs is allocated within a contiguous region, while the other jobs must be partitioned into two pieces and be allocated to regions that surround the first job? How to choose which job to allocate to a contiguous region, and which job to allocate to dispersed partitions? For the job that is partitioned, how to partition it into separate pieces? In other words, how should we cut a job and allocate it to disperse regions?

In addition to the communication rate that is used in Section 6.1, two more communication parameters are considered: *internal competing level* and *congestion factor*. The internal competing level of a path within a job is defined as the number of the path's starting and intermediate competing paths that belong to the same job. The average internal competing level of a job is the average of the internal competing

levels for all paths within the job. For the detailed explanation and examples of this parameter see Chapter 4. The congestion factor of a channel is defined as the number of competing paths that share a channel. In contrast to the internal competing level, the competing paths that compose the congestion factor may come from any job. This parameter is a measure of contention at a channel. To isolate the effect of parameters due to job interactions, we consider the following scenarios:

- Scenario A: One job has a higher rate of communication than the other job. Both jobs are composed of a 4-by-4 matrix-transpose pattern that has significant internal contention. Since the pattern of each job is the same, with this scenario we examine the effects of communication rate on job placement.

- Scenario B: One job has a 4-by-4 matrix-transpose pattern and the other job has a 4-by-4 diagonal pattern. Both jobs have the same rate of communication. In this case we isolate the effect of internal competing level on job placement. A difference between the jobs is that one has a significant amount of internal contention relative to the other.

6.2.2 Effect of Communication Rates

We use Scenario A to investigate the effect of communication rate on the placement of jobs on the MPC. Scenario A can be represented as the two jobs illustrated in Figure 6.7. Each job has the same communication pattern, but different rates of communication. We compare the effects on performance when the job in the contiguous portion of the system communicates at a lower rate in comparison to the communication rate of the job that is partitioned into two pieces and is to the outside of the first job. Likewise, we will examine the performance when the job to the inside communicates at a rate that is higher than the job to the outside.

As illustrated in Figure 6.7, the communicating paths of the job to the inside serve

Figure 6.

as interm

the job t

the effect

by the ef

inside job

intermed

Figur

outside j

x-axis re

of the o

Gray sh

each job

Ima

two figu

the com

the out

when t

rate of

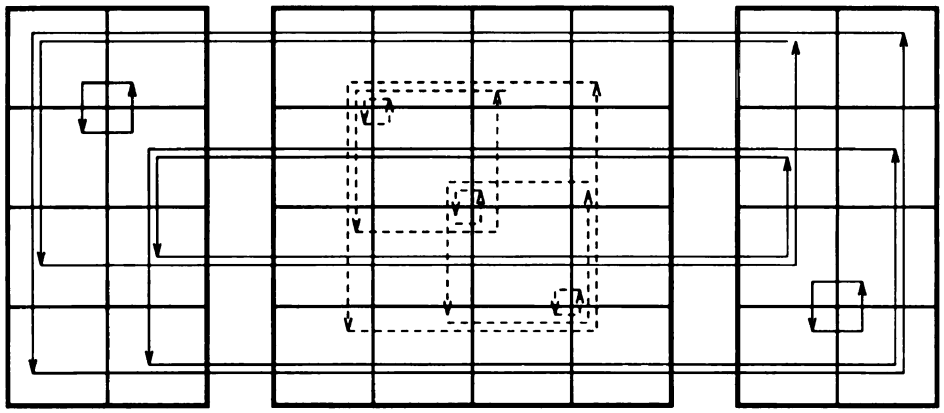


Figure 6.7. Job interaction model for investigating the effect of communication rate.

as intermediate contention points to the job that is placed to the outside. Likewise, the job to the outside will provide starting contention to the inside job. Therefore, the effect of the outside job on the performance of the inside job can be explained by the effect of starting competing paths on a channel. Similarly, the effect of the inside job on the performance of the outside job can be explained by the effect of the intermediate competing paths.

Figures 6.8 and 6.9 show the communication delays of the inside job and the outside job as a function of the MTBSs of two jobs, which range from 0 to 500. The x-axis represents the MTBS of the inside job, and the y-axis represents the MTBS of the outside job. The communication delays are represented as 2-D contour lines. Gray shadings in each figure represent different communication delay thresholds for each job at various communication rates of the pair of jobs.

Imagine a diagonal line is drawn from the point (0,0) to the point (500,500) for the two figures. The upper triangle due to the diagonal line would represent the case when the communication rate of the inside job is greater than the communication rate of the outside job. The lower triangle due to the diagonal line would represent the case when the communication rate of the outside job is greater than the communication rate of the inside job. For both figures, the contention delay represented in the upper

Figu
insid

Figure

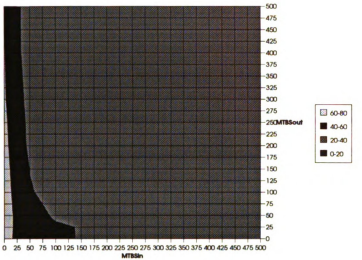


Figure 6.8. Communication delay of inside job: The x-axis represents the MTBS of inside job and the y-axis represents the MTBS of outside job.

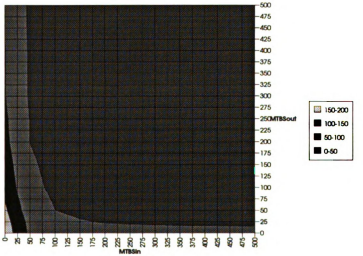


Figure 6.9. Communication delay of outside job: The x-axis represents the MTBS of inside job and the y-axis represents the MTBS of outside job.

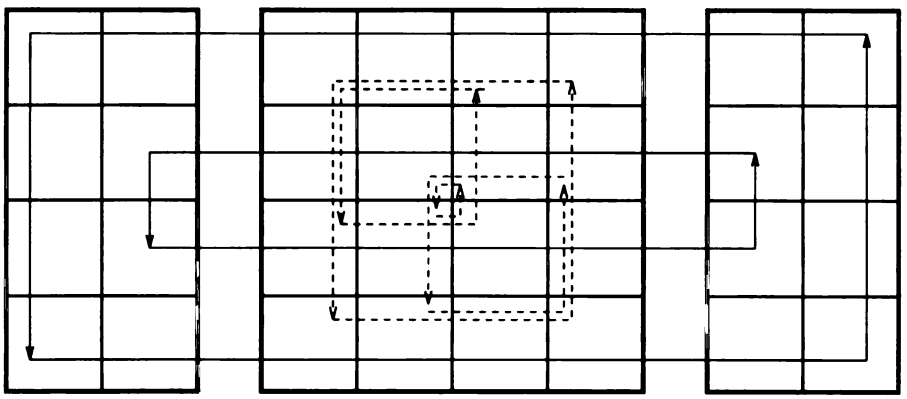
triangle is greater than the contention level in the bottom triangle. This means that it is more desirable for both jobs if the job that communicates less frequently is to the inside of the job that communicates frequently.

6.2.3 Effect of Internal Competing Level

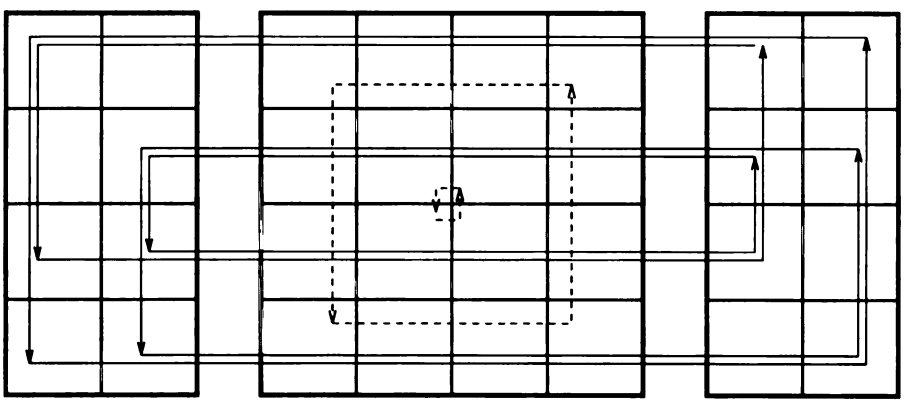
We use Scenario B to examine the effect of internal competing level on job placement when the two jobs have the same communication rates but have different communication patterns. The transpose job has a higher internal competing level in comparison to the diagonal job. We compare two ways to allocate the jobs. First, we allocate the transpose job to a contiguous partition and allocate the diagonal job to the outside with two pieces. Likewise, we consider the case when we reverse the method of allocation. Figure 6.10 illustrates the cases. We use the same MTBS for both jobs and partition the outside job in the middle and remove one of the communication path of the transpose job. This modification of removing one of the communication paths is made because we do not want to change the congestion factors of all channels in the inside job after exchanging the location of two jobs. Therefore, only the internal competing levels are exchanged.

Figure 6.11 compares the communication delays for the two cases. The communication delays are shown as functions of the communication rate. The solid lines represent the results when the matrix-transpose job is to the inside of the diagonal job, while the dotted lines represent the opposite case. The starred lines indicate the communication delays of the modified transpose job and the circle lines indicate the communication delays of the diagonal job.

The job located to the outside has higher communication delays than the job located to the inside, regardless of the communication patterns. This is because the intermediate contention delay caused by the inside job is much more severe than the starting contention delay caused by the outside job. The overall communication



Matrix-transpose inside—diagonal outside.



Diagonal inside—matrix-transpose outside.

Figure 6.10. Job interaction model for internal competing level.

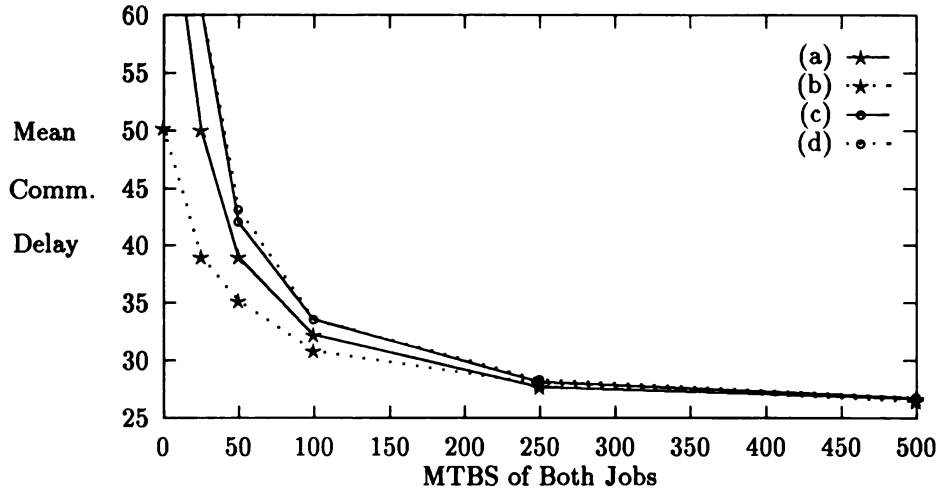
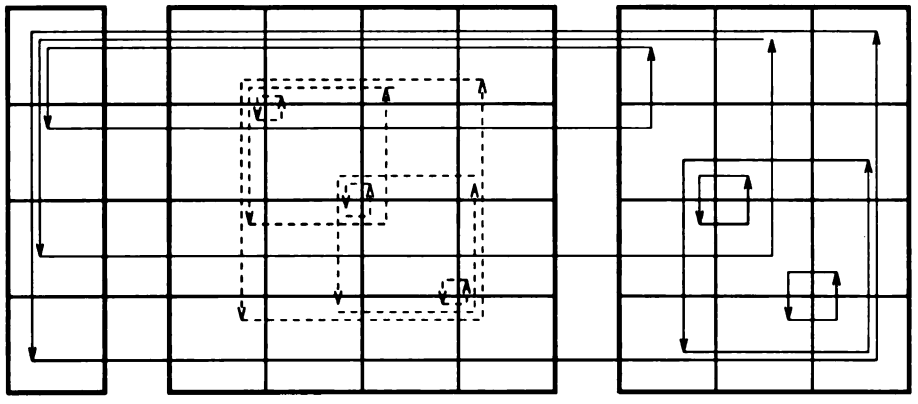


Figure 6.11. Effect of exchanging internal competing levels. (a) represents the communication delay of the inside job and (c) represents the communication delay of the outside job while the inside matrix transpose job interacts with the outside diagonal job. (b) represents the the communication delay of the inside job and (d) represents the communication delay of the outside job while the inside diagonal job interacts with the outside matrix transpose job.

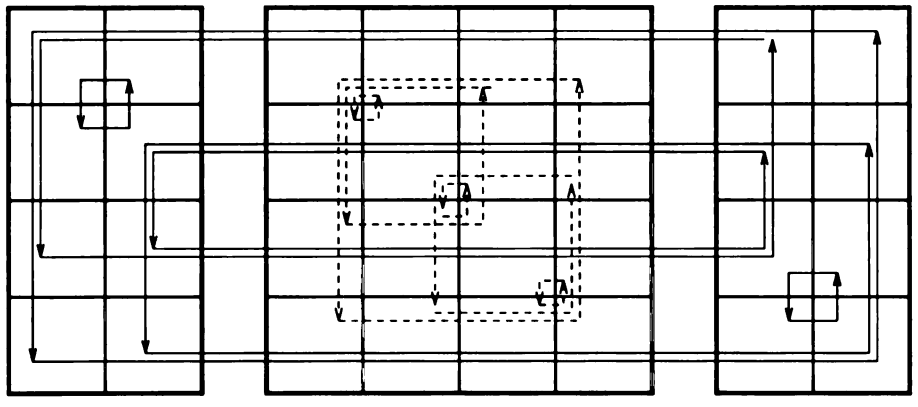
performance when the matrix-transpose pattern is to the outside of the diagonal job is better than when it is to the inside. This is because when the matrix-transpose is to the outside, the outside job creates a large number of starting competing paths for the inside job. The performance effect caused by the number of starting competing paths is not large. When the matrix-transpose job is to the inside, it causes a large number of intermediate contention paths for the outside job. The performance of a job is very sensitive to the number of intermediate contention points, and therefore the diagonal job to the outside will suffer significantly.

6.2.4 How to Partition: Effect of Congestion Factor

Suppose we have determined which job will be place to the inside and which job will be placed to the outside. We must also decide how to partition the outside job.



First partitioning method.



Second partitioning method.

Figure 6.12. Job interaction model for partitioning.

For simplicity, we focus our attention on partitioning along a vertical line. Different manners of partitioning will change the congestion factor even if the communication rates and the internal competing levels do not change.

We compare two different methods of partitioning using the model in Scenario A, which has two transpose jobs communicating with different rates. Figure 6.12 illustrates the two methods of partitioning. The first method partitions the outside job between the first and second columns. The second method partitions between the second and third columns. The first partitioning method causes one row to have a high congestion factor and three rows to have low congestion factors. The second

partitioning method causes each row to have a medium congestion factor.

Figure 6.13 shows the effect of the communication rates of the outside job on the communication delay of the inside job. The dotted line is for the first partitioning method, and the solid line is for the second partitioning method. The communication delay of the outside job for the first partitioning method is lower than the second partitioning method for all ranges of communication rates of the outside job. However, the communication delay of the inside job for the first partitioning method is lower than the second partitioning method only if the communication rate of the outside job is not too high. The reason is because the high congestion caused by the outside job saturates the channel when the communication rate is high. Therefore, if the communication rate of the outside job is not too high, then partitioning at the vertical line where the overall congestion factor is lower is a good decision.

6.2.5 Discussion

Let us consider the scattered processor allocator of a MPC system that serves multiple jobs simultaneously. When a job is scheduled for allocation by the job scheduler of the system, the processor allocator allocates the job in a contiguous partition, if possible. Otherwise, the allocator may partition the job into several pieces and allocate the sub-pieces into scattered regions. In this case, our conclusions of Section 6.2.2–6.2.4 could be used as "rules of thumb" by the scattered allocator. As an example, the result of Section 6.2.4 can be used for the Multiple Buddy Strategy in [7]. The strategy divides the request of the job into smaller square submeshes whose sides have sizes of 2^k when there are no contiguous partitions that are big enough for the required submesh. At each time of division, the scattered allocator should decide which among the four equal buddies is divided further. Obviously, different decisions cause different ways of partitioning the job. The result of Section 6.2.4 suggests that if the communication rate the job is not too high, then partitioning at the vertical line where the overall

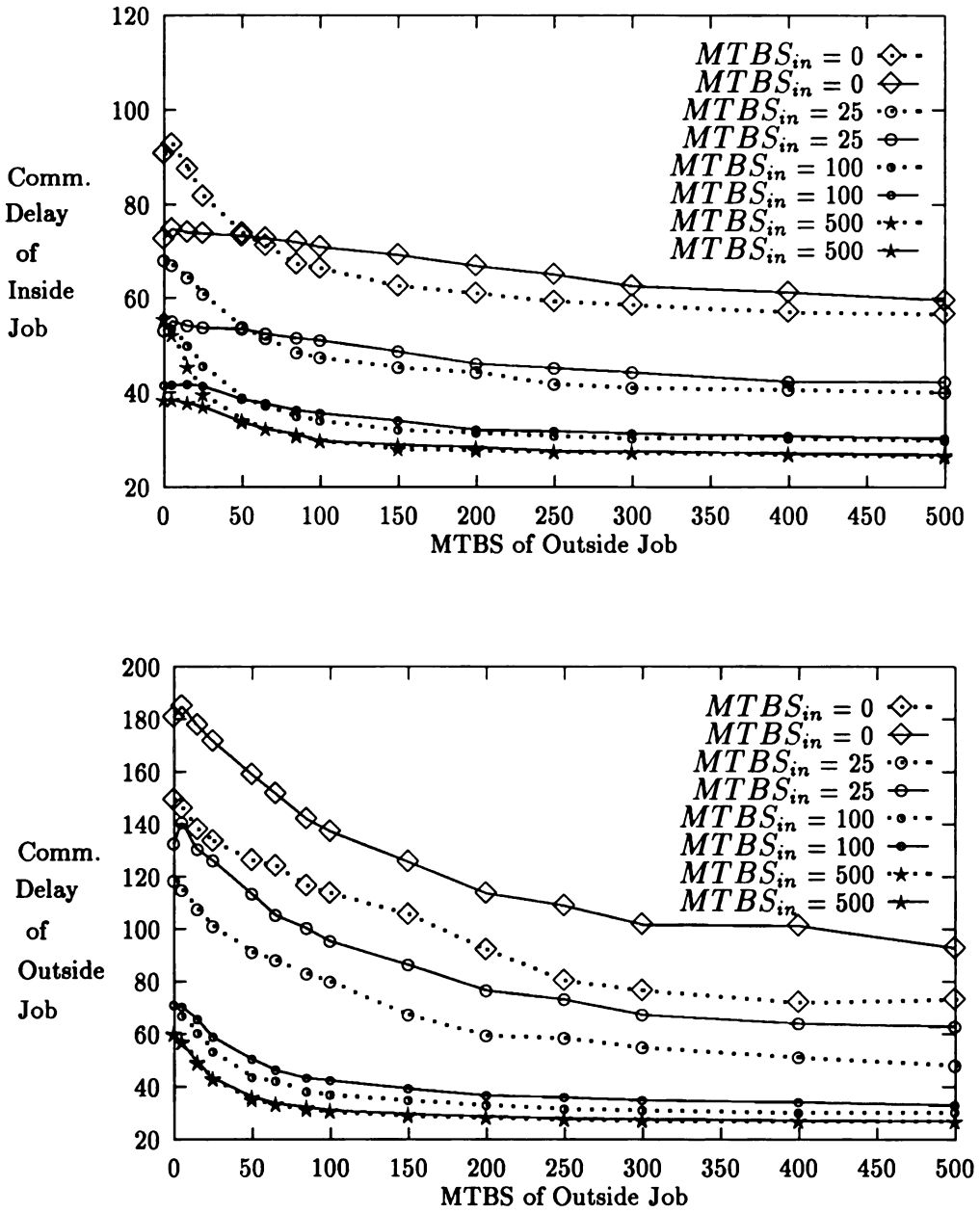


Figure 6.13. Effect of partitioning.

congestion factor is lower is a good idea.

As an second example, we can apply the results of Section 6.2.2 and 6.2.3 in order to improve the performance of the Naive allocation strategy that is used in [7]. Under the strategy, a job request for k processors is allocated the first k free processors in a row major scan of the mesh. According to the result of Section 6.2.3, the communication rate affects the communication interference between the interleaved jobs more significantly than the internal competing level. So, in the situation that the processors that surround the submesh allocated to a job should be allocated to the other job, the processor should check the communication rate of the already allocated job. If the communication rate of the allocated job is high, then the result of Section 6.2.2 suggests that it would be better not to allocate the processor to the job. The allocation surrounding a job with high communication rate may decease the communication performance of both jobs significantly. In this case, the allocator would improve the performance of the system by allocating the job to next possible places in a row major, or do not allocate the job in such a scattered way if there are no other possibilities.

An important argument against the practicability of our approach is that the characteristics of communication patterns and rates are not known before the time of processor allocation. If no characteristics of any job can be known in advanced, then no technique to exploit the characteristics can be used. Nevertheless, many jobs that use parallel processing systems execute for long periods and are re-executed many times. Many researchers have been building tools to analyze the performance bottlenecks of parallel computations. For jobs that execute for long periods, and are re-executed many times, the analysis of the patterns and rates of communication of the long and frequently executing jobs may be very beneficial to the overall performance of every user of the parallel processing system. Therefore, when these job characteristics are acquired, they might be used by the processor allocator when it makes its decisions

of scattered allocations.

6.3 Summary

Jobs interact with each other due to overlapping communication paths. The overlapped competing paths cause contention delays to be suffered by each job. Our results from an analytical model and simulation indicate that the effects of intermediate competing paths are more significant than the effects of starting competing paths as the number of competing paths and the communication rate increase. Our analysis provides guidelines for placing and partitioning jobs. For example, it is better to have the highly interactive jobs partitioned and placed at locations that cross the paths of other less interactive jobs, rather than partitioning the less frequently interacting jobs and placing them at locations such that they suffer from many intermediate contention points. Likewise, the point at which a job is partitioned is important, such that it is beneficial if the point of partitioning will lower the congestion factor.

CHAPTER 7

EFFECTS OF JOB SIZE IRREGULARITY ON DYNAMIC RESOURCE SCHEDULING

In order to provide a highly utilized parallel computing system, the problem of fragmentation has been addressed by a number of research studies. Most research has focused on developing innovative processor allocation strategies that can minimize system fragmentation. Many innovative strategies for allocating jobs to parallel computing systems have been proposed [66, 6, 67, 69, 7]. The proposed processor allocation strategies have been used in association with first-come-first-serve job scheduling, in which the jobs come to the system in a single queue. We will use SQ-FCFS to notate this job scheduling approach. A cited reason for using SQ-FCFS for job scheduling is because researchers wish to focus on the relative merits of the processor allocation strategies being explored. More importantly, SQ-FCFS is favored because it has an inherent notion of fairness, which is to say that jobs are served as they arrive in the single queue, and jobs are not favored on the basis of the size of the partitions that

they require.

Nevertheless, Krueger et al. [15] observed for a hypercube multicomputer system that the system performance is affected more significantly by the job scheduling algorithm rather than the processor allocation strategy. Sophisticated processor allocation strategies do little to improve the response time in relation to job scheduling strategies. By allowing jobs to be scheduled in an order that does not follow FCFS, the system utilization can be improved while the system fragmentation is reduced. Krueger et al. proposed the *scan* [15] discipline for hypercube multicomputers. According to the scan discipline, job arrivals are placed in multiple queues corresponding to the sizes of the subcubes requested. Jobs are scheduled by scanning through the non-empty queues, similar to the c-scan algorithm for disk scheduling [106]. The authors showed that with the scan algorithm a simple processor allocation strategy such as *buddy* allocation performs as well as more sophisticated strategies under most workload conditions for a hypercube system [15].

In contrast to the hypercube, jobs requesting resources on a general 2-D mesh system could request computing nodes that form irregular-shaped submeshes. In this chapter, we investigate effects of irregularity of job shapes and sizes and effects of a job scheduling strategy that uses multiple queues on the performance of dynamic scheduling. For the study, we examine a dynamic scheduling system that schedules jobs with requests that range from regular-shaped partitions of a multicomputer to irregular-shaped partitions. The employed job scheduling strategy, called the *BWQ-search algorithm*, provides additional opportunities for allocation of resources to jobs that require smaller submeshes when the job scheduler is blocked by a large job that cannot be assigned immediately. By means of this algorithm, we identify important components of a multiqueue job scheduling strategy and evaluate the relative benefit of the components. We address that irregularities of the shapes and sizes of jobs are important factors affecting the performance of a resource scheduling algorithm in a

2-D mesh multicomputer.

Section 7.1 provides motivation for our performance study of the job scheduling problem in a 2-D mesh system. A simulation model of a system on which our approach is analyzed is presented in Section 7.2. Section 7.3 includes a description of the job scheduling strategy. The results of a study of the performance is given in Section 7.4. Based on the simulation results, we discuss effects of the variability in the size and shape of an incoming job on the performance of the job scheduling algorithm. A summary and conclusions are given in Section 7.5.

7.1 Motivation: Irregularity of a 2-D Mesh System

The development of a dynamic resource scheduling algorithm for a general 2-D mesh multicomputer must deal with the inherent property of irregularity in the size of job requests and the irregularity in the shape of processor allocations. The irregularity of a 2-D mesh system is clearly illustrated, if it is compared with the inherent regular characteristics of jobs scheduled and allocated to a hypercube multicomputer system. Due to the special structure of a hypercube system, an incoming job request to the hypercube will require 2^k computing nodes that are configured as a subcube of the hypercube. Note that the unallocated parts of the system are also 2^k subcubes. It is straightforward to develop a job scheduler and a processor allocator that utilizes these regularities in order to reduce system fragmentation. In contrast to the hypercube, jobs requesting resources on a general 2-D mesh system could request computing nodes that form irregular-shaped submeshes. Figure 7.1 shows an example of the allocation of jobs to submeshes in a 2-D mesh multicomputer system. It assumes that the allocated submeshes match the shape and size requested by incoming jobs.

The gray rectangular regions represent the allocated processors and the white

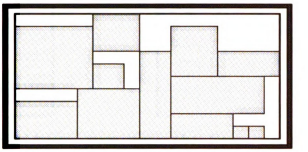
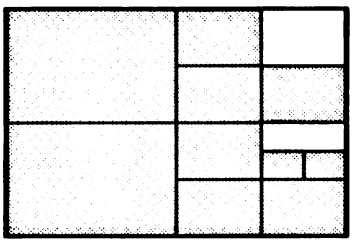


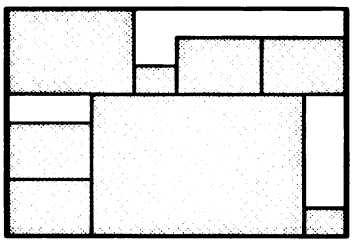
Figure 7.1. Processor allocation in a general 2-D mesh multicomputer system.

regions represent the unallocated regions of free processors. Most of the white regions have the irregular shapes and sizes and therefore it may be difficult to find a place to accommodate the next scheduled job even if there remains a partition for the job. The average amount of system fragmentation may vary and depends on the freedom that the shape of jobs may assume and sizes that a job can request. Greater freedom in the variations of shapes may cause larger system fragmentation. An important design issue for a resource scheduling strategy is to study the amount of performance degradation with respect to the amount of irregularity of the size of a job or the irregularity of the shape of the partition of processors the job will occupy. For this purpose, we study the performance effects of four representative cases of jobs, which can be described in terms of the shapes of the perimeter of the required partitions. The shapes are chosen with the restriction that processors are allocated in contiguous regions in which the routing technology of the multicomputer will not overlap one job's message traffic with the message traffic of other jobs. Then, the type of jobs illustrated in Figures 7.2(a)-(d) would not overlap message traffic of different jobs. For our study, the length of the sides of a partition will determine the shape of the jobs. The following job shapes are considered:

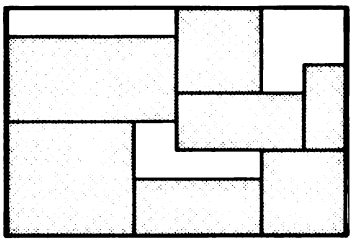
- *Square-2^k*: each side length is equal, and the length is 2^k, where $k \geq 0$.
- *Square-var*: each side length is equal, but can be any integer value in the range



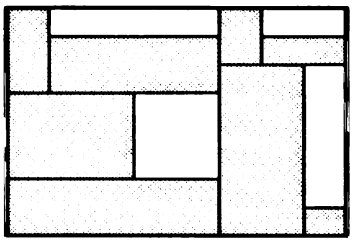
(a) Square-2^k



(b) Square-Variable



(c) Rectangular-restricted



(d) Retangular-unrestricted

Figure 7.2. Four different types of inputs.

from 1 to the maximum side length of the mesh system.

- *Rectangular-restricted*: side lengths of a job partition can be different, but the difference should not be larger than a given constant.
- *Rectangular-unrestricted*: each side length can be any integer value from 1 to the maximum size of a side of the mesh system.

7.2 Simulation Model

We model a dynamic scheduling system in which job arrivals follow a Poisson process. An incoming job consists of a number of interacting tasks that communicate via messages and specifies the geometry of a submesh it will need to occupy. The job will be allocated to a submesh of the requested geometry to avoid internal fragmentation.

The four cases of jobs described in the Section 7.1 are considered. The hold time for each job is assumed to be exponentially distributed and independent.

Since little information is available about the computing time demands required for jobs on mesh multicomputers, we consider the case that jobs are executed on a multicomputer in order to increase the throughput produced by a job, which is described as *uncorrelated* workloads by [15, 34]. Therefore, the amount of work done by each processor is independent of the submesh size. A job that consumes a large submesh will hold the submesh relatively the same length of time as a job requiring a small submesh. The larger submesh will have a greater computational throughput.

Our model of a multicomputer system consists of a processor manager and a number of general-purpose processors that are interconnected by a 16*16 2-D mesh network. In our study we consider only the allocation of processors to a job in a contiguous region. In order to allocate processors to a job in a contiguous region, we used the frame sliding method[6], which was developed for a general 2-D mesh system and allocates jobs in contiguous regions in an efficient way of minimizing system fragmentation. For evaluating components of job scheduling algorithms, a group-based strategy, called *BWQ-algorithm*, is used and described in the next section. The algorithm is studied for its ability to increase system utilization by reducing system fragmentation and improve the mean job turnaround time.

7.3 The BWQ Searching Algorithm

One method for restricting fragmentation due to processor allocation is to group jobs of similar sizes to locations in the multicomputer in a close vicinity. It is straightforward to implement the concept of grouping within a job scheduler by using multiple queues. Depending on how the queues are manipulated, an implementation may be classified as either *blocking-multiqueue scheduling* or *nonblocking-multiqueue schedul-*

ing. If a job is so large that it cannot be assigned to adjacent free processors, the first strategy blocks the job scheduler until processors become available. In contrast, the second strategy provides an additional opportunity to allocate jobs that require smaller submeshes. The job scheduler searches for jobs within the current queue or other queues. The underlying idea behind blocking-multiqueue scheduling is to take advantage of the grouping effect by blocking the job scheduler and collecting jobs of the similar sizes during the blocking period. This strategy can help reduce system fragmentation. However, as the blocked job waits, not only does its turnaround time increase but the turnaround times of other waiting jobs increase, which may result in a large average job turnaround time for the system.

We propose a nonblocking multiqueue-based job scheduling algorithm, called the BWQ algorithm. This job scheduling algorithm is divided into two parts: the *Between Queue (BQ) policy* and the *Within Queue (WQ) policy.* The BQ policy controls the order that a scheduler selects queues. The WQ policy controls the order that jobs are selected from a queue. Each queue has its own *Within Queue(WQ) policy.* In general, the WQ policy for each queue reorders the jobs within the queue so that in some situations a smaller job can be allocated before a larger job for which a partition is not available. However, the WQ policy is simply FCFS if all jobs in a queue have the same size.

7.3.1 Between Queue (BQ) Policy

The BQ Policy controls the order that queues are considered for selecting jobs to allocate. Queues are ordered according to the size of the jobs they contain, ranging from the smallest to largest jobs as illustrated in Fig. 7.3.

Initially, the scheduler searches the queue that holds the smallest jobs and schedules its jobs according to the WQ policy of the queue. If the queue is empty, the scheduler moves to the next non-empty queue in circular-right pattern and tries to

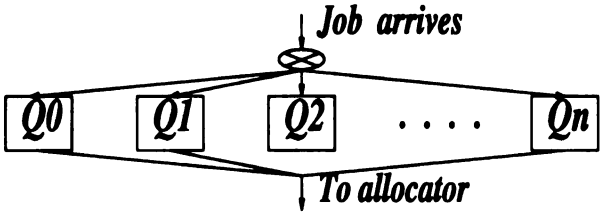


Figure 7.3. Multi-queue based job scheduling.

schedule the jobs of the queue with the WQ policy. When all jobs are scheduled and assigned without blocking, the BQ policy moves to the next non-empty queue. If no jobs within the queue can be scheduled for allocation, the job scheduler moves back to a non-empty queue that holds smaller jobs. The scheduler is not necessarily blocked from assigning jobs for which partitions exist. Therefore, an additional opportunity for resource allocation is given to smaller jobs that require smaller submeshes. This non-blocking property may decrease the turnaround times of the smaller jobs and reduce the external fragmentation in the system. Depending on the scheduling policy, the scheduler may move to the queue with the smallest jobs or to the previous non-empty queue. The policy needs a limit on the number of times the scheduler can return to queues holding smaller jobs to avoid starvation of larger jobs. Every time the scheduler passes a large job, the priority of the job is increased by 1. If the priority reaches the *Between Queue Search Limit (BQSL)*, then the scheduler is blocked to schedule the "starving" job. Initially the priority of each job is set to zero. The BQSL is a design parameter with performance implications that we will discuss later in this section.

7.3.2 Within Queue (WQ) policy

The WQ policy is controlled by a *lookahead window* that provides an additional allocation opportunity for smaller jobs within a queue. When a job or multiple jobs are ready to be scheduled within a single queue Q_j, the jobs that are chosen from the

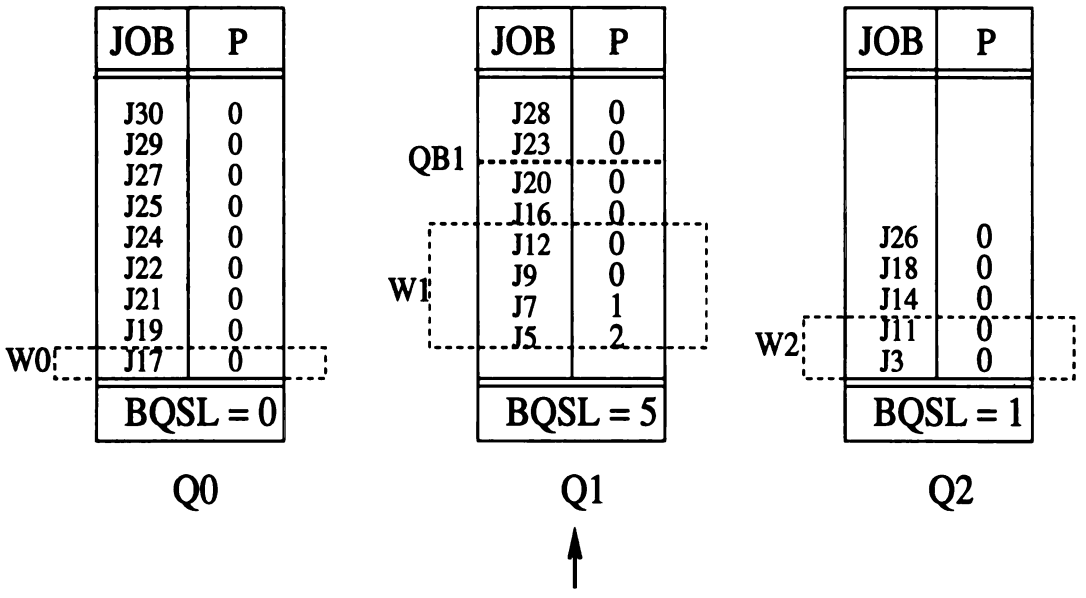


Figure 7.4. An illustration of job scheduling of three queues.

queue for scheduling come from the set of jobs within the *lookahead window* of the queue. Initially the set of jobs in the lookahead window are the n jobs at the head of the queue, where n is the lookahead window size. If n is greater than one and the first job of the queue cannot be allocated, then the job scheduler considers jobs in the queue within the lookahead window to find a job that is allocatable. If it finds such a job, it schedules the job and looks to the next n jobs. Otherwise, the job scheduler moves to another queue according to the BQ policy. A large lookahead window size increases the probability of finding a job that can be scheduled, but also increases the cost to search for a job. The window size needs to be bounded depending on the type of jobs assigned to a particular queue. The window size of each queue is a design parameter that is discussed later in this section.

Figure 7.4 shows the state of a queuing system of an job scheduler that has 3 queues. The first queue (Q0) is a FCFS queue, i.e., the size of the lookahead window is 1 and the BQSL is fixed to 0. Jobs in the queues have priority P set to 0. Notice that $J5$ and $J7$ in queue Q1 have $P > 0$, which means that these jobs were not able

to be scheduled in previous attempts.

The second and third queues, Q1 and Q2, are general lookahead window queues. Each has a BQSL given as a design parameter. After scheduling the jobs in Q0, the scheduler is currently working in Q1. Some jobs have been assigned and the scheduler searches within the lookahead window of Q1. Notice that Q1 has a dotted-line that is called the *Qboundary*. This parameter is discussed in the next subsection.

7.3.3 Design Parameters

In this subsection, three design parameters are studied to maximize system performance. To isolate the effect of each parameter, we fix the other parameters at specific values. Experiments were conducted for all four job types. Since some of our studies produce similar results, only the results of the rectangular-unrestricted input are displayed when they are similar.

Number of queues

The number of queues used is an important design parameter that can have a significant effect on the system performance. In a hypercube multicomputer system, the number of queues used is the number of possible subcubes whose sizes are powers of 2 [15]. For the 2-D mesh system we cannot apply the same method since jobs have irregularity. This parameter depends on factors such as the allocation strategy, job arrival rates, distribution of job sizes and system size.

Figure 7.5 presents the effect of increasing the number of queues on the performance of BWQ job scheduling algorithm in our simulation model of mesh system. We could use a large number of queues due to the variability of the possible sizes for all types of jobs. However, the number of queues was limited to six or less in order to study the effect of increasing the number of queues. The simple case of using only one queue implies a FCFS queue. The BQSL of each queue is set to 30. Thresholds that

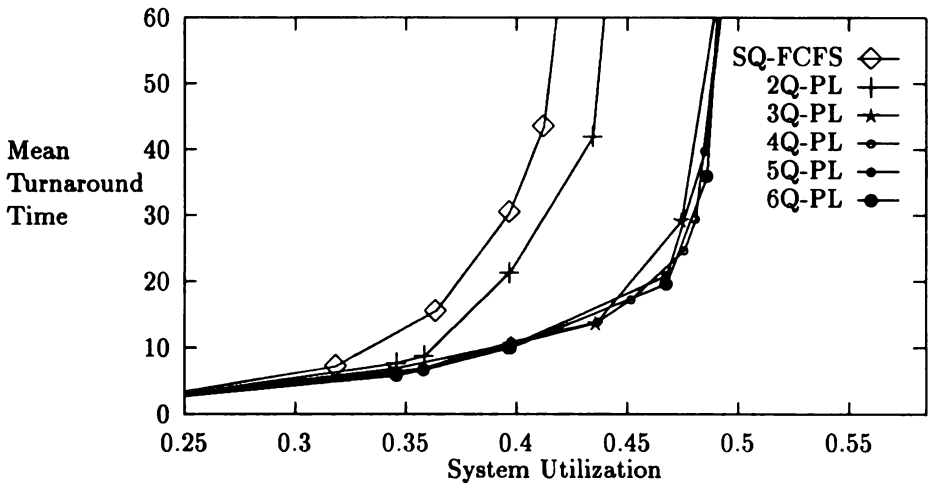


Figure 7.5. Response Time (RT) effect of increasing number of queues on the performance of job scheduling policy: BQSLs = 30, Input type = Rectangular-unrestricted.

establish the sizes of jobs placed in each queue are defined in order to balance the number of jobs that arrive at each queue. When the number of queues is increased from one to three, the mean turnaround time decreases significantly. However, for more than 3 queues the mean turnaround time does not improve. Consequently, for a certain number of queues we obtain a system performance that is near the maximum. The appropriate number of queues may change, depending on the allocation strategy, distribution of job sizes, and the size of the mesh system. Larger mesh systems with incoming jobs that have greater variability in sizes may need more queues. For our simulation model, 3 queues are enough.

Between Queue Search Limit(BQSL)

Another important design parameter is the BQSL of each queue. Any queue with a BQSL equal to zero becomes a scheduling bottleneck. Large jobs in a queue that cannot be scheduled immediately block all other jobs in the system, which may result

in an increased mean turnaround time. Therefore, the BQSL of each queue should be tuned in relation to the BQSLs of the other queues. Results providing information about the tuning of the BQSL are displayed in Fig. 7.6 and 7.7. Three queues were used and incoming jobs of the unrestricted rectangular type were generated with the arrival rate set to result in 50% system utilization. Because Q0 is designed as a FCFS queue, the BQSL for Q0 was set to 0. The simulation results show that mean turnaround times are much higher when at least one of the BQSLs is zero. Therefore, a significant improvement in performance occurs when the BQSL of each queue is larger than zero. These results imply that providing an additional scheduling opportunity for smaller jobs when blocking would occur (i.e., non-blocking) has a profound effect on the mean turnaround time. However, it is inappropriate to set the BQSL to a value that is too large. If the BQSL is greater than or equal to 5, then the mean turnaround time increased significantly since large jobs can wait too long. Another interesting aspect is the relative effect of the BQSL of one queue on the performance of the jobs in another queue. When we compare Fig. 7.6 and 7.7, the effect of the value assigned to the BQSL of Q2 is more significant than the value assigned to Q1.

Lookahead Window

The lookahead window of each queue is used to reorder jobs within a queue. The size of the lookahead window is another factor that should be tuned appropriately for system performance. At a high level of system utilization, as illustrated in Fig. 7.8, large lookahead windows improve the performance. The diamond plot of Fig. 7.8 shows the simulation results when the lookahead window size of queues 1 and 2 is one (WS=1). The cross plot of Fig. 7.8 shows the improved performance for larger window sizes. We have found that the performance improvement due to the lookahead window is sensitive to the type and characteristics of the incoming job stream. An incoming job stream with a wide variation in the sizes of jobs benefits much more

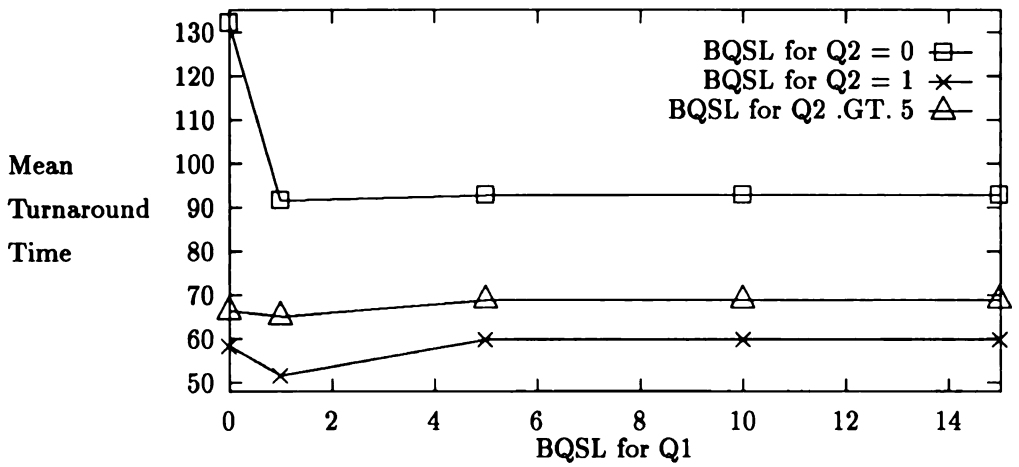


Figure 7.6. Effects of changing BQSL of Q1: 3 Queues with both window sizes = 1, Input type=Rectangular-unrestricted.

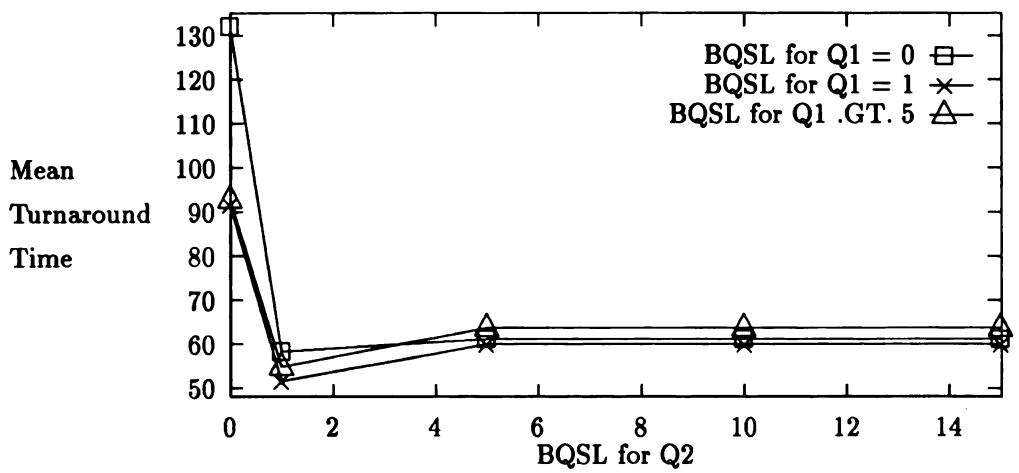


Figure 7.7. Effects of changing BQSL of Q2: 3 Queues with both window sizes = 1, Input type=Rectangular-unrestricted.

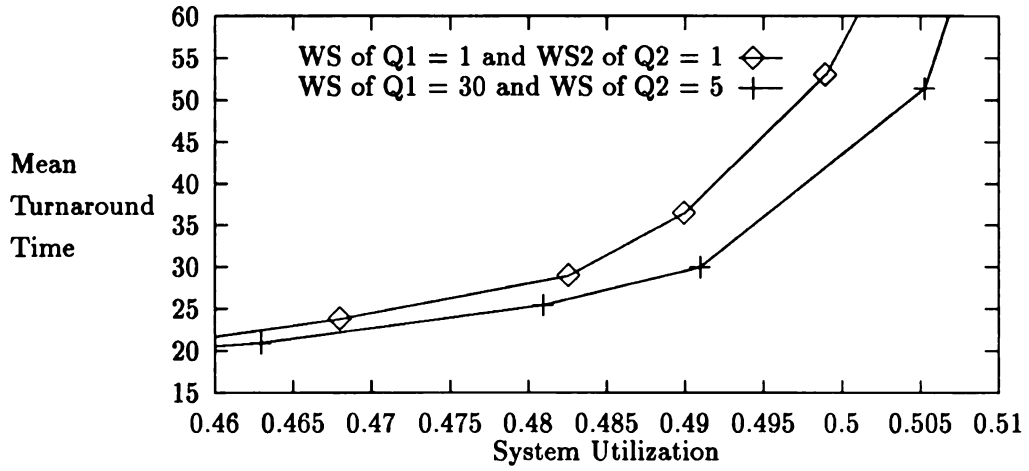


Figure 7.8. Response Time (RT) effects of lookahead windows: 3 queues, BQSLs of Q1 and Q2 = 30, Input type=Rectangular-unrestricted, FS-allocator.

from a larger lookahead window size than an incoming job stream with uniform sizes of jobs.

Qboundary

The purpose of the boundary is to avoid starvation of jobs. Only the current queue for which the scheduler is selecting jobs has a Qboundary. It is established when the scheduler first considers a queue and remains until all jobs below the Qboundary have been scheduled. Jobs arriving after the Qboundary has been established will not be considered candidates during the current scheduling phase of the queue. Suppose there is no Qboundary established for a queue, e.g., Q1. Further suppose all new jobs are placed in Q1. Therefore, the scheduler will only select jobs from Q1, which results in the other jobs starving.

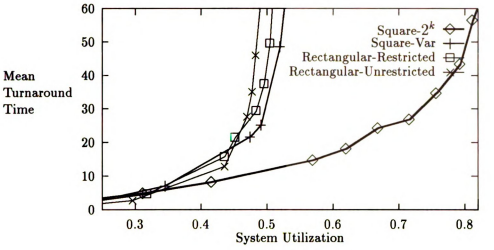


Figure 7.9. Response Time (RT) effects of variability in size and shape on job turnaround times of BWQ algorithm: Q1 with BQSL = 10 and Q2 with BQSL = 1.

7.4 Performance Effects of Variability in Size and Shape

Once we obtain parameters for the BWQ algorithm, we are ready to examine the effects of the variability of sizes and irregularity of shapes of jobs on the dynamic scheduling system. Figure 7.9 shows results that demonstrate these effects. The four curves show the mean job turnaround times of BWQ scheduling algorithm for the four different types of inputs described in Section 7.2. An important feature is the difference between the curves for Square-2^k input and the other inputs. The mean job turnaround time of Square-2^k is stable up to 60% system utilization. In contrast, the other inputs become unstable before 45% system utilization. This fact implies that a variability of input sizes has a dramatic effect on the performance of a job scheduling policy.

One of the key factors that contributes to the difference in the results is the system

fragmentation. As the system utilization increases to 80%, the external fragmentation due to Square-2^k input decreases to 12%. However, the other types of inputs cause the system to become unstable at lower system utilizations, resulting in a high external fragmentation (38% external fragmentation at 45% utilization). Although our job scheduling algorithm tries to reduce system fragmentation by allowing non-blocking for the other types of inputs, the processor allocator has difficulty in assigning jobs efficiently because the sizes of inputs vary greatly. While the variability of input size has a profound effect on the performance of job scheduling algorithms, irregular input shapes do not make a significant difference in the system performance, as illustrated in Fig. 7.9. The performance for the input types Square-var, Rectangular-Restricted, and Rectangular-Unrestricted are similar as the system utilization increases. As a result, we have the freedom to change input shapes when assigning jobs. Nevertheless, the "less-regular" jobs (i.e., Rectangular-Unrestricted) display a little poorer result than the "more regular" jobs (i.e., Square-var). From the above analysis, we can conclude:

- The type of input has significant effects on the performance of a job scheduling algorithm. A regular-shaped job – regular-shaped (2^k) cluster – can be scheduled with significantly better performance in comparison to other input shapes that have less regularity.

- If the geometry of job partitions are allowed to be arbitrary shapes, then differences between separate classes of irregular jobs (*e.g.,* Square-Var, Rectangular-Restricted, Rectangular-Unrestricted) are insignificant.

Restructuring Job Input

Suppose we allow scattering a submesh requested by a job into several smaller submeshes. This assumption is possible if wormhole routing is used as a switching tech-

nology for the mesh multicomputer and the frequency of communication between processors allocated to a job is not large. We can improve the performance of the job allocation strategy by dividing the required submesh into square-shaped submeshes with side lengths equal to $2^k, k \geq 0$. A simple buddy processor allocation strategy with a multiqueue-based job scheduling algorithm can be used to achieve good job scheduling performance. However, further complications are required in order to co-schedule the separate submeshes that are part of a single job. Also, if a large amount of communication is required between the processors allocated for a job, then the contentions for the communication system can be significant. As a result, the communication contention generated by one job will likely affect the performance results obtained by other unrelated jobs. Other jobs can be inhibited from accessing the communication network if jobs must communicate between processors that are dispersed in the multicomputer. For jobs that require a large amount of communication, it is advantageous to allocate partitions of processors to the jobs in contiguous regions of the multicomputer so that they will not inhibit the performance effect of other jobs. Therefore, we need a strategy that can transform the shape of an incoming job to allow for efficient utilization of the system, while keeping the submesh allocated to the job in a contiguous shape.

7.5 Summary

We discussed the performance effects of arbitrary sizes of jobs in 2-D mesh system and proposed a job scheduling strategy based on a concept of grouping. The job scheduling algorithm used is a nonblocking multiqueue-based job scheduling algorithm that is efficient and suitable for general 2-D mesh multicomputer systems. In order to decrease the job turnaround time and increase the system utilization, the strategy minimizes the external fragmentation by reordering the scheduling of jobs.

The simulation results showed that the amount of inherent system fragmentation of a 2-D mesh system depends on the amount of irregularity of the sizes and shapes of job requests. There is a large performance improvement between very regular-shaped partitions and the other types of partitions. The results were analyzed in the context of developing a processor allocation strategy for wormhole-routed 2-D mesh systems.

CHAPTER 8

EFFICIENT JOB SCHEDULING WITHOUT DISCRIMINATION AGAINST LARGE JOBS

Research activity has been divided between those seeking better system response times by means of job scheduling and those who insist that fair scheduling requires FCFS. Since SQ-FCFS does not favor small jobs to the detriment of large jobs, researchers continue to use SQ-FCFS and concentrate on processor allocation techniques. Nevertheless, other researchers have made efforts to develop job scheduling algorithms that can achieve significant performance gains by reducing system fragmentation. The BWQ-search algorithm in Chapter 6 is the effort in this direction. Generally, these algorithms favor small jobs by changing the execution order and have been criticized because they provide improved average performance by favoring small jobs at the expense of large jobs [65].

In this chapter we propose a new job scheduling discipline, called the *HELM* discipline, which takes advantage of the performance gains that are possible via job scheduling, while it ensures that small jobs are not inappropriately favored in comparison to large jobs. The HELM discipline adapts its scheduling policy to the changes

of workload by using parameters that dynamically update their values depending on the history of the workload (A detailed description of these parameters is given in Section 8.2). Under low loaded conditions HELM follows the SQ-FCFS discipline. If the load is low, little performance gain would be possible even if a non-FCFS scheme was used. Under a highly loaded condition, the HELM discipline reorders the jobs to increase the utilization of the system. However, to avoid the long waiting time charged to large jobs that have been skipped several times in order to allocate smaller jobs, the discipline checks the waiting time experienced by a job when it considers scheduling it. If a job waits for such a long time in comparison to the average waiting time experienced by jobs that have been served, then the discipline raises the priority of the job so that it will not be skipped again. To evaluate the fairness of the HELM scheme, we compare the performance ratio that large jobs achieve relative to the performance of small jobs. This is evaluated by classifying jobs into two groups, large and small, and then computing the ratio of average waiting time experienced by the large jobs to the average waiting experience by the small jobs. We compare our algorithm's ratio with that of SQ-FCFS.

The rest of the chapter is organized as follows. The following section describes the HELM discipline. The design parameters of the discipline are examined in Section 8.2. Section 8.3 presents simulation results and a comparison to other disciplines. Concluding remarks are given in Section 8.4.

8.1 The Proposed Strategy

The HELM scheduling discipline controls the order of scheduling by using three types of queues: High-priority queue, Entrance queue, and Lookahead Multiple queues (HELM).

- **The Entrance Queue (EQ)**: The entrance queue is a FCFS queue with a lookahead window (LW). This queue is used for keeping jobs in the order of arrival. When an incoming job arrives at the multicomputer system, the job is placed in the entrance queue in the FCFS fashion. The job scheduler schedules jobs from EQ one by one. When an attempt to allocate the current job has failed, the HELM discipline allows the scheduler to look ahead to the next job, passing the current job to the Lookahead Queues (LQ), which will be described below. The amount of look-ahead is controlled by the lookahead window (LW), whose size is adaptive to the change of workload. Tuning the size of LW for the entrance queue is discussed in Section 8.2. Due to the adaptive lookahead window, the HELM discipline has a tendency to follow the FCFS discipline, especially under low loaded conditions. Under highly loaded conditions, however, the FCFS discipline is inefficient because of the randomness in the shapes and sizes of consecutive jobs. As the system workload becomes high, the HELM discipline places large jobs in the LQs where jobs are scheduled in an efficient order to achieve better system performance.

- **The Lookahead Queues (LQ)**: The jobs passed to the lookahead queues are classified according to the size of request submesh and are queued to one of the LQs in the order of arrival. The HELM discipline attempts to schedule all the jobs in the LQs one by one, skipping the jobs that cannot be allocated. The advantage of using lookahead queues is that the HELM scheduler can allocate as many jobs as possible under a highly loaded condition. However, jobs are scheduled in an different order from FCFS. To avoid unfair treatment of large jobs, the HELM discipline uses a parameter called the *Upper Bound of Waiting Time (UBWT)*. UBWT is a linear function of the average waiting time of the jobs that have been served so far. The coefficient of the linear function has to

be tuned. The UBWT is a dynamic parameter that is updated every time a job leaves the system after its execution. When a job in a lookahead queue cannot be allocated by the scheduler at its turn, the HELM discipline compares the waiting time of the job to the current value of UBWT. If the value of UBWT is larger, then the scheduler passes the job. Otherwise, the job is moved to the high-priority queue in which the job would not be passed again until it is allocated. The high priority queue is described below.

- **The High-priority Queue (HQ)**: The jobs that have waited in the system more than the UBWT are moved to the high priority queue. This queue is a typical FCFS queue. This queue is highest in priority in comparison to the other queues; the HELM scheduler checks this queue first when a job releases processors allocated to it. If there are jobs in this queue, the HELM scheduler schedules the jobs one by one in the FCFS fashion until the queue is empty.

Figure 8.1 describes the algorithm of the HELM discipline. The job scheduler based on the HELM discipline first checks whether the HQ is empty. If not, then the job scheduler schedules the jobs in HQ one by one in FCFS fashion until the HQ is empty. When the HQ becomes empty, the control of the job scheduler moves to LQs. If there are some jobs in the LQs, then the HELM scheduler considers all the jobs in LQs, checking whether each considered job can be allocated. After considering all the jobs in a LQ, the HELM scheduler moves to the next non-empty LQ. If the allocation of the current job is possible, then the job scheduler schedules the job. Otherwise, the scheduler checks whether the waiting time of the job is larger than UBWT, the upper bound of waiting time, where $UBWT = K * AWT$, K is the positive constant and AWT is the average waiting time of the jobs that have left after their executions. If the waiting time of the job is larger, then the job is move to HQ. Otherwise, the scheduler passes the job and considers the next job in the next non-empty LQ. After

```
do{
  wait( not_empty );
  while( there is something in queues ) {

    /* SCHEDULE THE HIGH-PRIORITY QUEUE (HQ) */
    while ( HQ is not empty ) {
      schedule and allocate the jobs in HQ in the FCFS fashion
    } /* end of while */

    /* SCHEDULE THE LOOKAHEAD QUEUES (LQ) */
    while( All LQs are not empty && All jobs in LQs are not considered ){
      J = the first non-considered job from the next non-empty LQ
      if (J is allocatable) then allocate J
      else {
        /* Update the Upper Bound of Waiting Time (UBWT) */
        /* The coefficient of UBWT, K1, is determined in Section 8.2.2 */
        UBWT = K1 * the average waiting time of the served jobs
        if( WT(J) > UBWT) move J to HQ
      }
    } /* end of while */

    /* SCHEDULE THE ENTRANCE QUEUE (EQ) */
    /* Update the Size of Lookahead Window (LookaheadSize) */
    /* The coefficient of LookaheadSize, K2, is determined
    /* in Section 8.2.3 */
    Cnt_Lookaheaded_Jobs = 0;
    LookaheadSize = K2 * the average queue length of EQ

    while(  EQ is not empty && Cnt_Lookaheaded_Jobs < LookaheadSize  ){
      J = the first job in EQ
      if (J is allocatable) then allocate J
      else {
         Cnt_Lookaheaded_Jobs++
         move J to one of LQ depending on the given classification policy
         /* In our simulation we classify the jobs
            depening on the size of job request */
      }
    } /* end of while */

    /* TO AVOID BUSY WAITING */
    if ( Nothing could be allocated ) then wait until a job leaves

  } /* end of while */

}while(TRUE);
```

Figure 8.1. The HELM Algorithm

considering all the jobs in LQs, the control of the HELM scheduler moves to the EQ. The scheduler considers jobs within the size of lookahead window. If the job can be allocated, then it schedules the job. Otherwise the job is moved to one of the LQs.

8.2 Design Parameters

We conducted a set of discrete event-driven simulations in order to find suitable values for the parameters of the HELM algorithm. The HELM discipline has three parameters to be tuned: the number of lookahead queues, the coefficient of UBWT for LQs, and the size of the lookahead window. To isolate the effect of each parameter, we varied the value of one parameter at a time while the other parameters were fixed to constants.

The simulator, which is written in the CSIM simulation language, progresses its time according to the occurrences of events, which are job arrivals and departures. The interarrival time and the service time of jobs are assumed to be exponential. In the simulation, the mean service time was fixed to 20 time units, but the mean interarrival time, called IATM, varied in order to generate various loads. When a job arrives to the system, the incoming job requests a number of processors. The number of processors requested by a job follows a uniform distribution. Given the request by a job, the processor management part of the operating system assigns the smallest submesh that can accommodate the job request. This method of submesh assignment generates submeshes having more uniform sizes than the method that was used in [68, 67]. For the simulation, a 32x32 mesh system (the total number of processors are 1024) was used. The processor allocation strategy used in the simulations is First-Fit strategy [68]. The simulations run a series of batches of 1000 jobs until a 95% confidence interval is achieved.

Four performance metrics have been employed for examining the performance of

the scheduling disciplines. First, the *job turnaround time* and the *system utilization* are measured, which are the most important performance indicators for users and system administrators. The job turnaround time, denoted by TAT, is defined as the time interval from the point when a job arrives in the system to the point when the job leaves the system after its service is done. The system utilization, denoted by UTIL, is defined the ratio of the number of allocated processors to the total number of processors in the system. Both are measured when a job leaves the system and the measured values are averaged after the simulation is done. As another indicator of system performance, the external fragmentation is measured. This metric will be denoted by FRAG. When the job scheduler is not able to allocate any jobs, the ratio of the number of idle processors to the total number of processors in the system is measured and averaged to compute the external fragmentation of the system. This metric is also used in order to explain the effects of reordering jobs on the system performance. Finally, to examine the fairness of the disciplines the L/S Ratio is measured. This metric classifies the jobs into two groups, a large group and a small group, depending on the submesh sizes assigned to the jobs. The median job size is used to distinguish to which group each job is classified. The ratio of the waiting time of the jobs in the large group to the waiting time of the jobs in the small group is defined as the metric, L/S Ratio. If this ratio is near to 1, the discipline is said to be fair.

8.2.1 Number of LQs

The two lines in the first figure of Figure 8.2 show the effects of the number of queues on the turnaround time (TAT) of the MQ-SCAN discipline and of the HELM discipline, respectively. MQ-SCAN is the job scheduling algorithm using Multiple Queues in which jobs are scanned similarly to the c-scan algorithm for disk scheduling [106]. This algorithm is a variation of the "scan" algorithm in [15], that is orig-

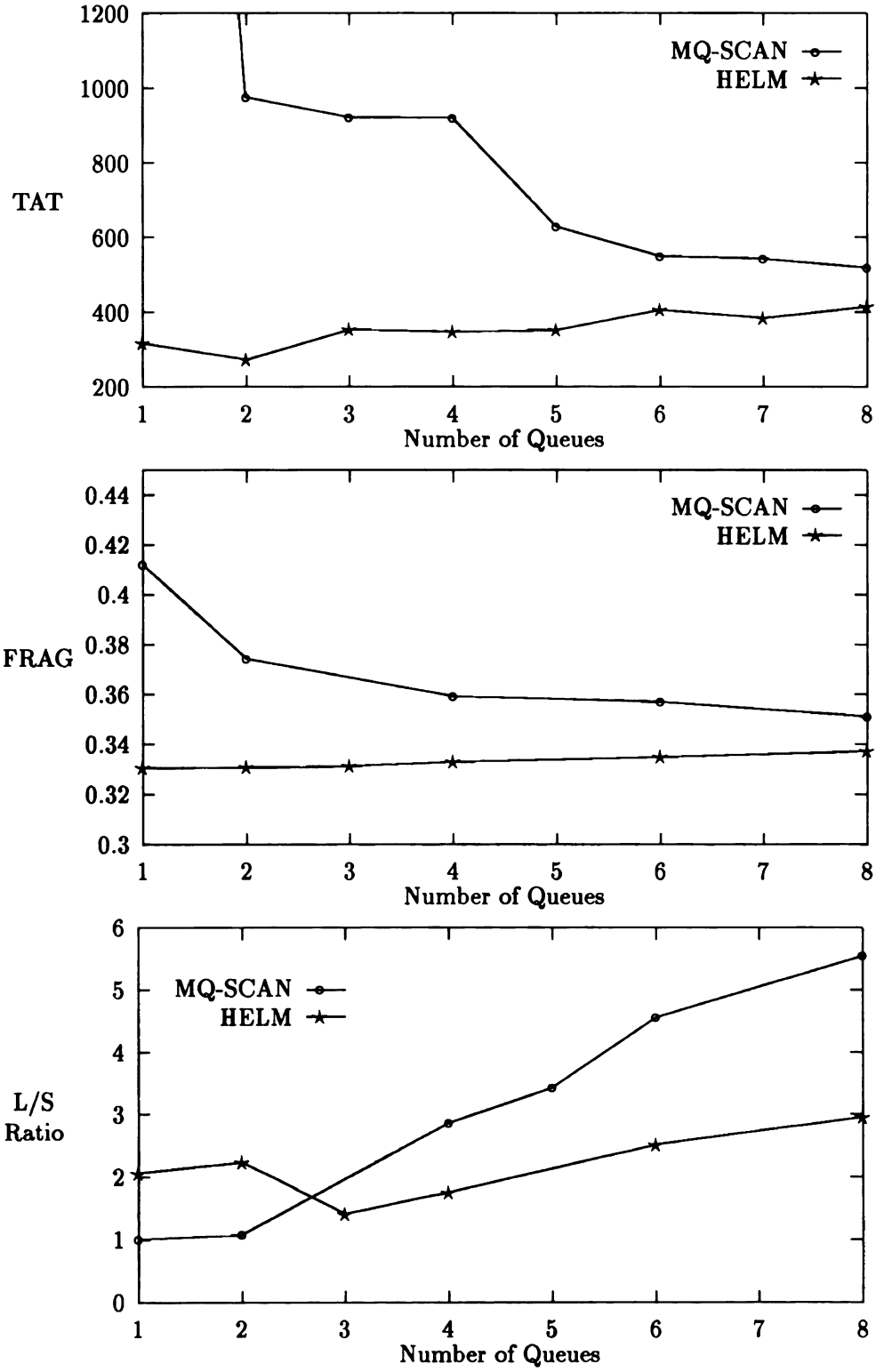


Figure 8.2. Tuning the Number of Queues in the HELM Algorithm.

inally designed for hypercube multicomputers. Like the HELM discipline, the MQ-SCAN discipline also employs multiple queues. In order to make the TAT sensitive to the changes of the number of queues, a high workload is applied.

According to the figure, as the number of queues increases the performance of the MQ-SCAN discipline improves significantly, while the TAT of the HELM algorithm is relatively low and steady. The improvement of the performance of MQ-SCAN can be explained in terms of the system fragmentation illustrated in the second figure in Figure 8.2. As the number of queues increases in the MQ-SCAN discipline, a greater number of small jobs can be considered for allocation. However, it also means that if more queues are used, then large jobs may be treated unfairly. Therefore, the L/S Ratio of MQ-SCAN increases significantly, as presented in the third figure of Figure 8.2. In contrast, as the number of the LQs increases, the system fragmentation of the HELM algorithm remains steady. Because the HELM algorithm considers all jobs in LQs for scheduling, a small number of queues enables the HELM scheduler to allocate a job if an allocation exists. For the remainder of the simulation study, the number of queues for MQ-SCAN is fixed to 5 and the number of LQs for HELM is fixed to 2.

8.2.2 Coefficient of UBWT

The first figure in Figure 8.3 shows TAT as a function of the coefficient of UBWT. UBWT is computed by multiplying the coefficient with the average waiting time of the completed jobs. The y-axis represents TAT, and the x-axis represents the coefficient. In the simulation, the number of lookahead queues is set to 2 and the size of lookahead window is set to one-half of the average length of EQ. The coefficient varies from 1 to 8. As shown in the figure, TAT changes significantly when the coefficient varies from 1 to 4. After 4, however, the coefficient does not have much effect on the performance of the system. If the coefficient is 1, most jobs in the LQs are moved to HQ and the

HELM discipline behaves as the FCFS discipline. Therefore it shows high system fragmentation and low L/S ratio. As the coefficient increases from 1 to 2, the role of the LQs increases. The coefficient 2 ensures that on average the job waiting time of the jobs in HQ is not much larger than two times of the average job waiting time. If the coefficient is larger than 2, the UBWT becomes too large and most jobs remain in LQs without going to HQ. For the remainder of the simulation study, the value of this coefficient is fixed to 2.

8.2.3 Coefficient for the Size of Lookahead Window

Figure 8.4 shows the effect of the coefficient of the size of lookahead window (LW) on the job turnaround time. When the current job in EQ cannot be allocated, the HELM scheduler moves the job to one of the LQs. To adapt to the change of the system workload, the size of LW is not fixed to a constant. Rather, the value could change depending on the number of jobs waiting in the EQ. For the simulation, the size of the LW is computed by multiplying the average queue length of EQ with the coefficient. In the figure, the coefficient varies from 0.1 to 1. As the rate increases to 0.5, the system performance is improved. The HELM scheduler with a very small lookahead window behaves as a SQ-FCFS. As the window size increases, the role of LQs become more active, but the L/S Ratio increases. If the window size is too large, the searching time in LQs increases, which decreases the system performance. For the remainder of the simulation, this coefficient is fixed to 0.5.

8.3 Simulation Results and Comparison

We compare the HELM discipline to two other job scheduling disciplines: SQ-FCFS and MQ-SCAN. The L/S Ratio of the SQ-FCFS algorithm is used as a standard of fairness. MQ-SCAN is the job scheduling algorithm using Multiple Queues in which

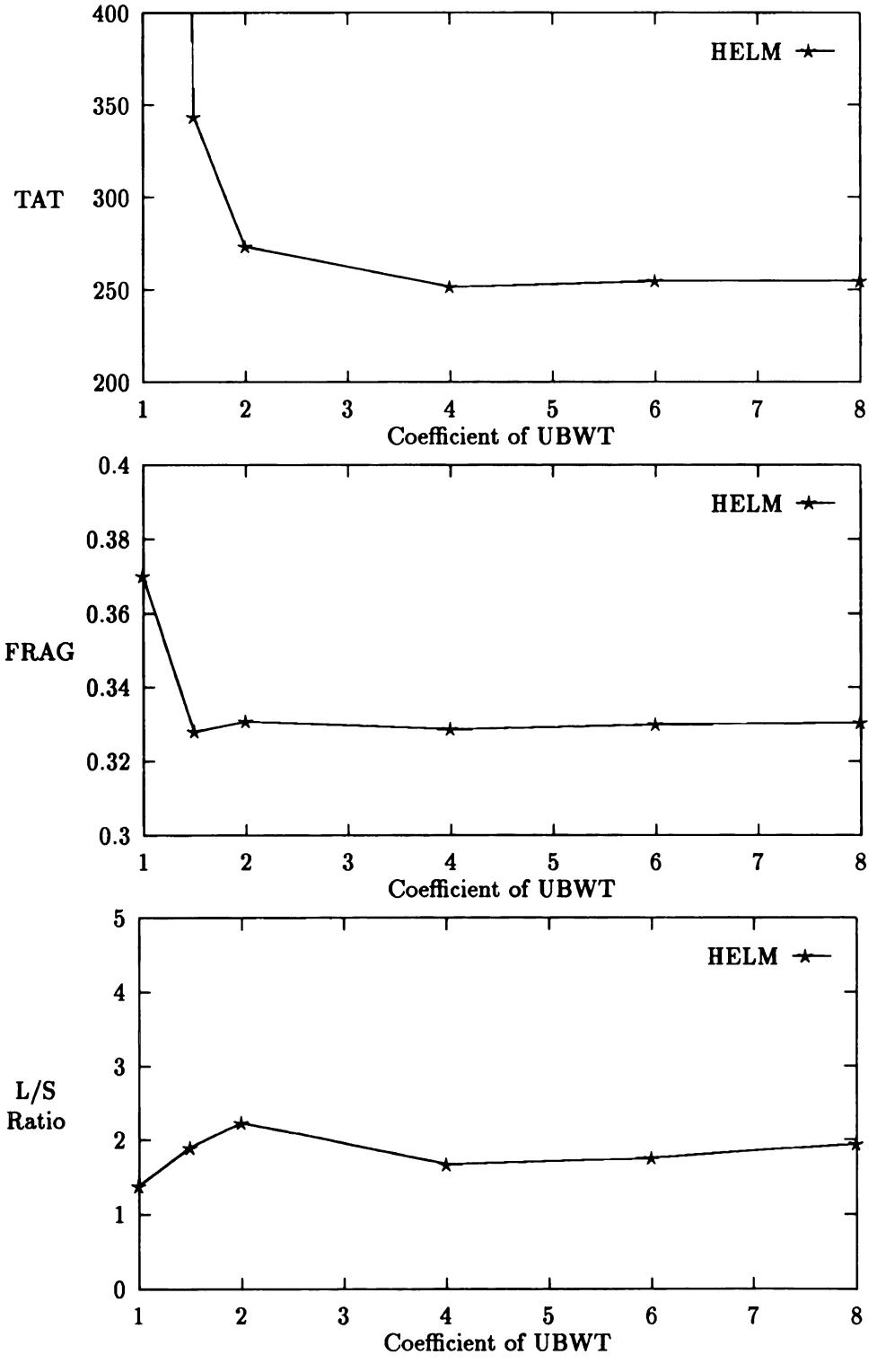


Figure 8.3. The Coefficient for the Upper Bound on Waiting Time.

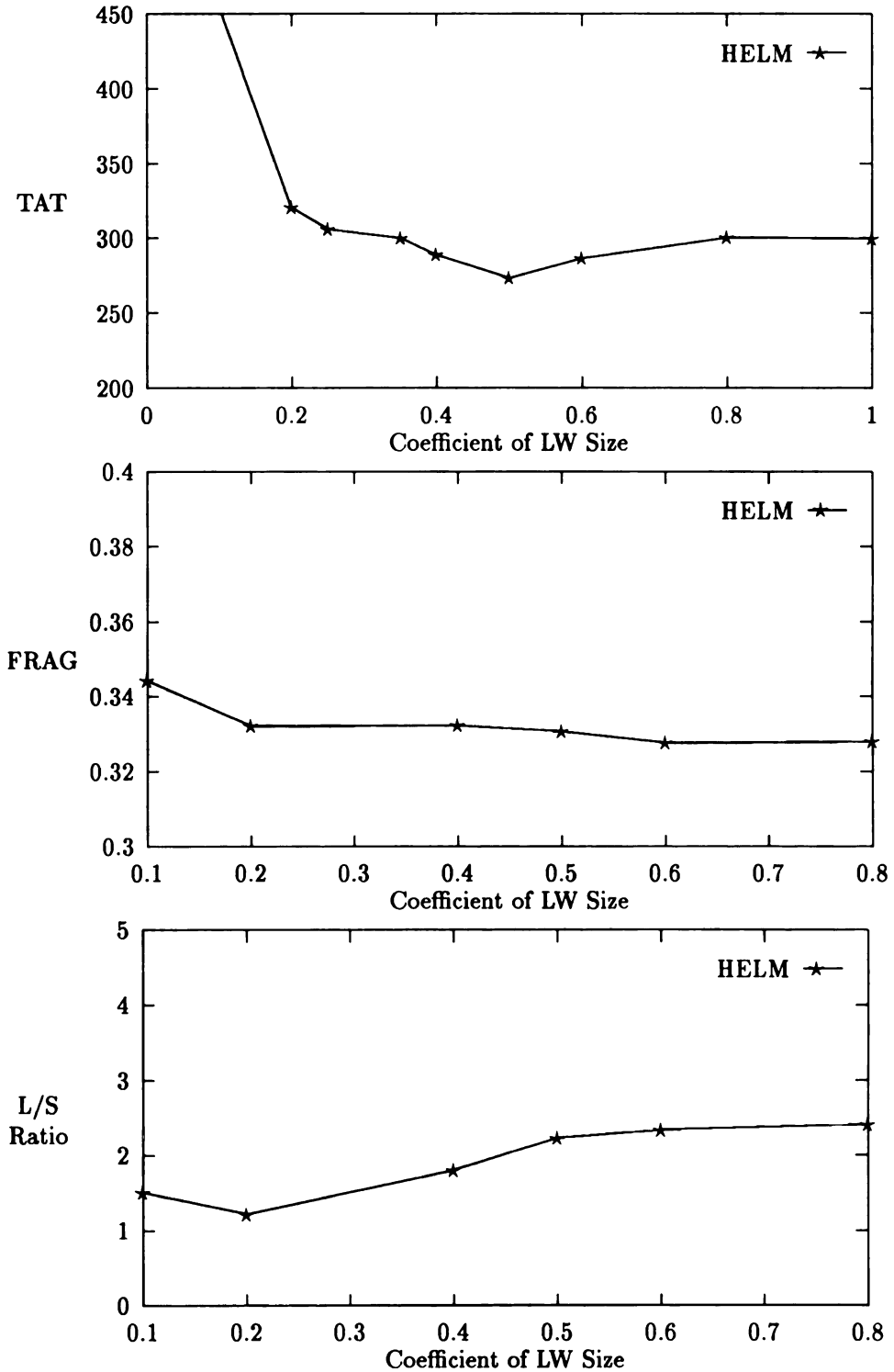


Figure 8.4. The Size of the Lookahead Window.

jobs are scanned similarly to the c-scan algorithm for disk scheduling [106]. This algorithm is a variation of the "scan" algorithm in [15], that is originally designed for hypercube multicomputers. Like the HELM discipline, the MQ-SCAN discipline also employs multiple queues.

8.3.1 Job Turnaround Time and System Utilization

Figure 8.5 presents the TAT of the scheduling disciplines for varying system workloads, which is controlled by the IATM (mean interarrival time). The TAT of the SQ-FCFS algorithm increases very quickly at higher loads. Compared with the SQ-FCFS algorithm, the multiple queue algorithms have much better TAT at higher loads. The HELM algorithm shows a little higher turnaround time than the MQ-SCAN algorithm under low loaded conditions, however its performance under highly loaded conditions is much better than that of MQ-SCAN. The reason why HELM shows a little higher turnaround time than MQ-SCAN is that it needs to maintain a steady L/S Ratio.

Figure 8.6 shows the system utilization of each algorithm. It can be seen that under various loads the HELM algorithm shows higher system utilization than the other algorithms. It means that the HELM algorithm schedules jobs more efficiently so that more jobs can be allocated on the system.

8.3.2 External Fragmentation

Figure 8.7 shows how the external fragmentation changes as a function of IATM. In the SQ-FCFS discipline, 41% of processors are idle no matter what is the system workload. In contrast, the external fragmentation of MQ-SCAN and HELM decrease as the workload increases. At very high load, the external fragmentation of MQ-SCAN and HELM are 36% and 33%, respectively. This tendency of decreasing external

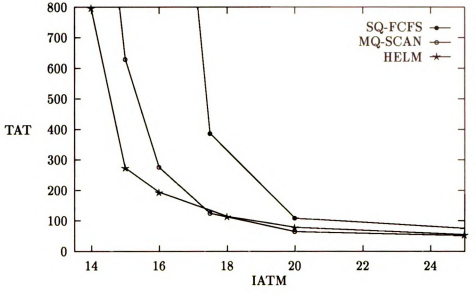


Figure 8.5. Comparison of the Turnaround Time for the Three Disciplines.

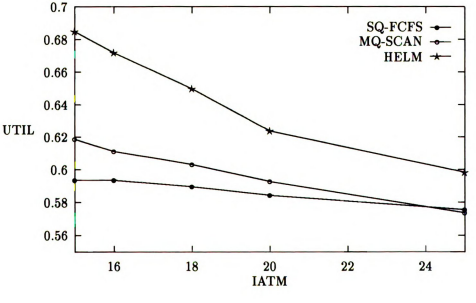


Figure 8.6. Comparison of the System Utilization.

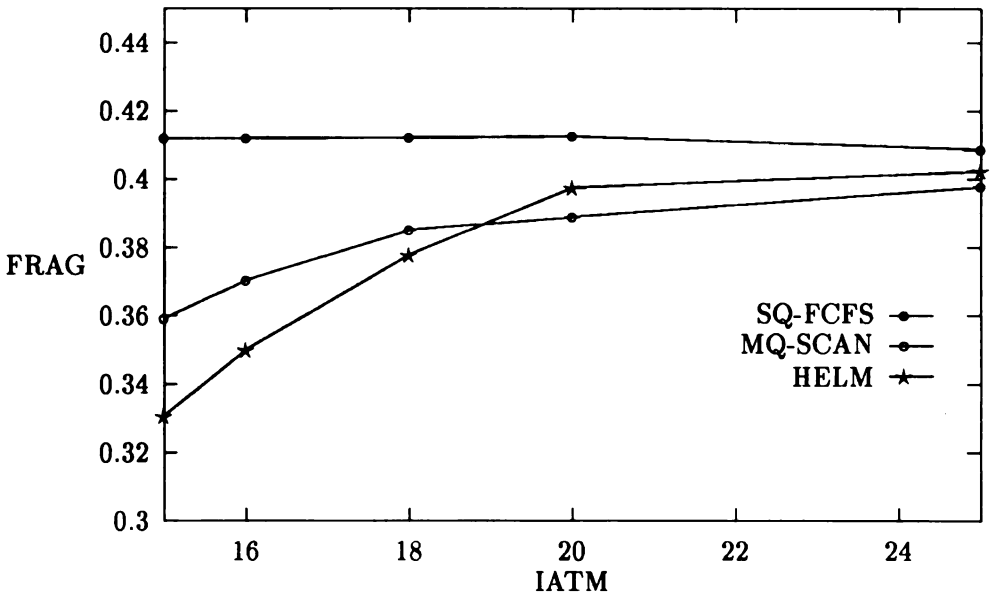


Figure 8.7. External Fragmentation.

fragmentation under high system workload means that the non-blocking aspect of the schemes makes the job scheduling algorithms more adaptive to the change of system workload. The HELM algorithm shows some external fragmentation under low loaded conditions. This is because the size of the lookahead window in EQ dynamically changes depending on the workload. Under low loaded conditions, the lookahead window has a small size such that the HELM scheduler only "looks-ahead" to a few jobs. Under highly loaded conditions, the size of lookahead window increases as the length of EQ increases. Therefore, more jobs move to LQs and the HELM scheduler allocates as many as possible. The HELM scheme can achieve high system utilization and low external fragmentation under highly loaded conditions.

8.3.3 Fairness

Figure 8.8 shows L/S Ratio of the three algorithms. The ratio of SQ-FCFS is near to 1 because SQ-FCFS does not discriminate on the basis of job size. In contrast, MQ-SCAN shows high L/S Ratio for many workloads. This is because the MQ-

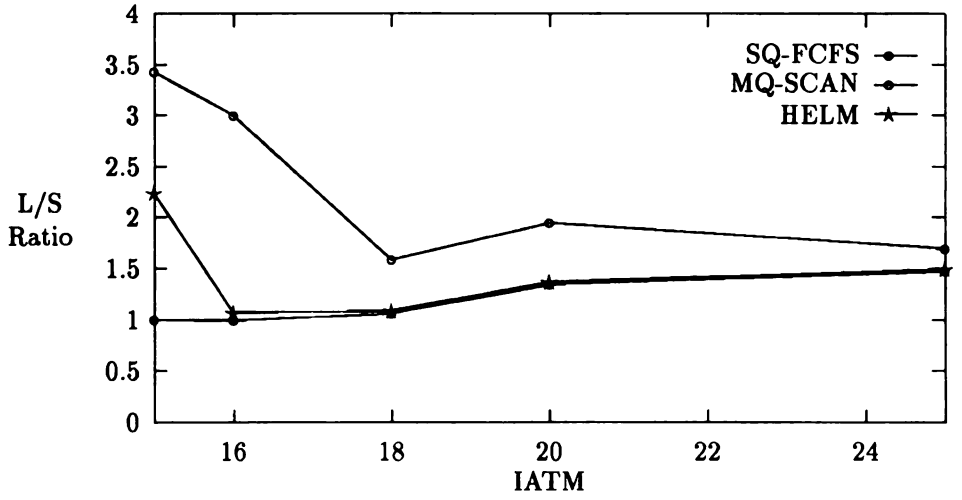


Figure 8.8. The L/S Ratio for the Three Schemes.

SCAN discipline changes the order of jobs to decrease external fragmentation. As MQ-SCAN, the HELM algorithm rearranges the order of job scheduling due to LQs under highly loaded conditions to achieve high system utilization. However, the HELM scheduler always checks its waiting time when a job is passed. If the waiting time of the job is larger than UBWT, then the scheduler moves the job to HQ so that the job would not be passed again. Because the value of UBWT dynamically changes depending on the system workload, HELM is more adaptive to the change of workload. Therefore, under highly loaded conditions the HELM scheme shows a lower L/S Ratio in comparison to the MQ-SCAN. Unlike MQ-SCAN, the HELM discipline shows the low L/S Ratio as SQ-SCAN under low loaded conditions. This is a very important performance characteristic of the HELM discipline.

8.4 Summary

We proposed in this chapter an innovative job scheduling discipline, called HELM, which adapts its scheduling policy to the changes of workload. HELM achieves low

job turnaround time and high system utilization while not inappropriately favoring small jobs to the detriment of large jobs. HELM manipulates three types of queues: the entrance queue, the lookahead queue and the high-priority queue. For jobs that arrive at the entrance queue that cannot be allocated, HELM moves them to the lookahead queues. The lookahead queues are non-blocking for allocation efficiency. If a job cannot be allocated, HELM searches for other jobs until it finds a job that can be allocated. For fairness and adaptiveness to workload, HELM changes the size of lookahead window of the entrance queue depending on the length of the entrance queue. Under low loaded conditions, this lookahead window will be small so that most jobs are scheduled in a FCFS fashion by the entrance queue. As the workload increases, the window size increases. Therefore, under highly loaded conditions, many jobs move to the lookahead queues and HELM can schedule them efficiently by reordering the jobs. The reordering of jobs in the lookahead queues decreases the system external fragmentation and increases the system utilization. However, to avoid the situation where a job waits in the lookahead queues too long, HELM uses a upper bound of waiting time in the lookahead queues. If a jobs waits longer than the bound, HELM moves the job to the high-priority queue. The high-priority queue keeps and schedules the jobs in the order of arrival time one by one. The upper bound of waiting time is a dynamic parameter whose value changes depending on the characteristic of the system workload. According to our simulation results, the HELM discipline shows much better performance under highly loaded conditions than the MQ-SCAN and SQ-FCFS disciplines. HELM schedules fairly as the SQ-FCFS discipline under low and medium loaded conditions.

CHAPTER 9

CONCLUSIONS AND FUTURE RESEARCH

Processor management is one of the important services provided by an operating system for massively parallel computers that serve multiple jobs simultaneously. Research activity on the processor management problem has been divided between those seeking better system utilization by allocating a job to any set of processors regardless of their geometric location (scattered scheme) and those who insist contiguous allocations in which each subpartition is a contiguous region of the MPC (contiguous scheme). In this thesis, we studied the processor management problem on a wormhole-routed 2-D mesh multicomputer for the issues that may be critical in both processor management schemes: the network contention issue and the system fragmentation issue. The major objective was to gain insight into the system behavior and to understand the basic principles underlying the performance of processor management strategies in the system. In this chapter, we summarize the major contributions made by this research and present the directions for future research.

9.1 Summary and Major Contributions

The effects of contention in wormhole-routed networks can cause the characteristics of one job to affect the performance of another job when a job is allocated to scattered partitions on an MPC. In this thesis, the performance degradation due to job interactions has been intensively studied. We studied the effects of competing paths on network contention due to the nature of wormhole routing networks when several paths are overlapped that have different communication rates. We proposed a general contention model, called the heterogeneous multipath contention model. The model is a representation of arbitrarily overlapped communication paths of jobs that have different message communication rates. Based on the proposed contention model, we analyzed the performance characteristics of 2D mesh multicomputers under contention due to job interactions. The analysis may help the system designer to gain insight into system behavior, to understand the basic principles underlying the performance of system strategies, and to compare scattered allocation strategies with contiguous allocation strategies.

As a starting point for analyzing the interactions and interference that occur between jobs, we examined interactions and interference between communication paths that have the same communication rate. We proposed two metrics that can be used when one wants to measure internal and external contentions between jobs in a multicomputer. These metrics are called the starting contention level and intermediate contention level. Based on the metrics, the internal contention delays of a stair-layered pattern and a complex transpose pattern were predicted and verified by means of simulation. According to the results, as the starting contention level increases, the communication increases for a wide range of communication rates. The amount of increase in communication delay depends on the rate of communication as well as the contention delays facing the external paths that contend at the starting point. Nev-

ertheless, the detrimental delays due to the intermediate contention level primarily occurs only at high rates of communication.

We also analyzed the performance degradation due to the sharing of network resources by multiple independent interacting jobs in a 2-D mesh wormhole-routed multicomputer system. The analysis is based on a divided-and-conquer strategy, which derives the communication delay at each contention point on a path. This strategy reduces a heterogeneous multipath contention model at each contention point to a heterogeneous 2-path contention model. The computation of the reduced model distinguishes the starting contention point from the intermediate contention points. In addition, we analyzed a contention model in which two jobs have complex internal communication patterns that overlap. These results help us understand the effect of job interactions on network performance such that we are better prepared for finding solutions to the problem of allocating processors in 2-D mesh wormhole-routed multicomputers.

Based on the performance analysis of network contention, we developed efficient processor allocations strategies for 2-D mesh wormhole-routed multicomputer systems. Our examination on the effects of the overlapped competing paths on the network contention indicated that the effects of intermediate competing paths are more significant than the effects of starting competing paths as the number of competing paths and the communication rate increase. This phenomenon is substantial in wormhole routing networks due to the nature of the wormhole routing switching technique. Based on this examination, we investigated the effects of several communication parameters on communication interference between interleaved jobs due to job partitioning and placement. Our conclusions drawn from simulation results provide guidelines for placing and partitioning jobs. For example, it is better to have the highly interactive jobs partitioned and placed at locations that cross the paths of other less interactive jobs, rather than partitioning the less frequently interacting

jobs and placing them at locations such that they suffer from many intermediate contention points. Our results also show that the overall communication delay is very sensitive to the relative partitioning and placement of jobs. This research offers just a starting point for more in-depth research to address placing and partitioning jobs. It emphasizes that careful partitioning and placing jobs in scattered allocation can reduce the negative performance effect of job interaction.

Another investigation of this thesis is to characterize the effect of job size irregularity. We examined a dynamic scheduling system that schedules jobs of various shapes from regular shape to irregular shape. According to the results, job's regularity in both shape and size can contribute to improved system performance. We found that the performance is similar when the system schedules jobs that request various types of irregular-shaped partitions. A large improvement in performance occurs if all jobs scheduled on the multicomputer request very regular-shaped partitions. This study gives us some insight into what are the important characteristics to be considered to design a system partitioning scheme for a processor allocation strategy. We outlined an approach for restructuring incoming job requests to use partitions from a multicomputer so that the performance advantage of regular-shaped partitions is utilized.

Finally, we proposed a new job scheduling strategy that can achieve low job turnaround time and high system utilization while not inappropriately favoring small jobs to the detriment of large jobs. Many innovative schemes for allocating jobs to parallel computing systems have been proposed in order to achieve highly utilized parallel computing systems. Since most schemes that have been proposed for allocating jobs to parallel computing systems have concentrated on approaches for processor allocation, the schemes have used First-Come-First-Serve (FCFS) as the job scheduling discipline. However, it has been previously established that job scheduling algorithms for parallel computing systems can have a large impact on the system utilization and

job response time. Schemes that use multiple queues, which reorder the sequence of jobs allocated to the parallel system, can be very effective in improving the system performance. However, such non-FCFS schemes have been criticized because they provide improved average performance by favoring small jobs at the expense of large jobs. The proposed job scheduling strategy, called the HELM discipline, adapts its scheduling policy to the changes of workload so that it behaves in a FCFS manner under low loaded conditions, but exploits performance enhancing features of multiple queue schemes under highly loaded conditions. According to our simulation results, the HELM discipline shows low job turnaround time and high system utilization while not inappropriately favoring small jobs to the detriment of large jobs.

9.2 Future Research

The research presented in this thesis has focused on the network contention and system fragmentation issues that are fundamental in developing efficient processor management schemes. Future work relating to this research will include the following.

- Although the analysis presented in this thesis can be successfully applied to characterize the communication performance of wormhole routing under job interactions, it has a limitation that the routing scheme should be deterministic, such as XY routing. Multicomputer systems can employ adaptive routing schemes that adapt to dynamic changes of network condition. A system using an adaptive routing takes advantages of taking another path in the conditions such as while message traffic is heavy on one path or a faulty node exists on one path. Moreover, for scattered allocation schemes in which jobs are allocated to processors in any dispersed regions, adaptive routing may be an better alternative than deterministic routing. Future research will include the study of examining the performance of scattered processor allocation strategies under

adaptive routing scheme. Adaptive routing will have effects on partitioning and placing a submesh request to small subpartitions in the system.

- Virtual channel can be used to reduce network contentions. In the systems supporting virtual channels, scattered allocation will become more competitive than in the systems without virtual channels. Therefore, it would be interesting to extend our analytic formula for the systems that support virtual channels. This extension causes modifications in our formula. Consider a 2D mesh multicomputer that supports virtual channels. Assume that a physical channel is split into at least two virtual channels. In the system, intermediate contention delays no longer exist. Each intermediate contention delay in the original formula should be substituted with the delay that takes in sharing a physical channel by two paths through the supported virtual channels. This delay is a variable that depends on the system characteristics to support virtual channels and the message communication rates on the paths. The modification for computing the delay at the first contention point may be more complicated. Let n be the number of competing paths at the first contention point and m be the number of virtual channels per each physical channel. If n is smaller than m, the starting contention delay in the original formula should be substituted with the delay in sharing a physical channel by the n competing paths through the supported m virtual channels. Again, the delay is a system-dependent variable. If n is larger than m, a contention might occur at the first contention point. This delay includes two kinds of delay: the delay in sharing a physical channel by the n competing paths through m virtual channels and the contention delay by $(n - m)$ competing paths. Research issues are how to compute the system-dependent delay and how to combine it with the contention delay generated by the $(n - m)$ competing paths.

- External fragmentation in 2-D mesh multicomputers occurs when an scheduled job fails to be allocated to every available subpartitions in the system. The failures can be caused by either types of mismatching: *size mismatching* or *shape mismatching*. Size mismatching occurs when the number of processors in an available subpartition is smaller than the number of processors required by the incoming job. External fragmentation due to size mismatching is unavoidable in general parallel processing environment where the sizes of incoming jobs vary and processor migration [64] and limit allocation [65] are not supported. Shape mismatching occurs when an subpartition has sufficient processors, but has a shape into which the submesh required by the job cannot fit. This type of mismatching occurs because any shape of submesh can be assigned to the job. It has been shown in this thesis that the regularity of the shapes of jobs is an significant factor affecting the performance of a resource scheduling algorithm in a 2-D mesh multicomputers. Another goal for future research is to design a set of rules for submesh shaping that can be applied in the process of submesh assignment and to analyze its effects on the performance of processor allocation strategies.

- Better processor allocation strategies that can achieve lower job turnaround time and higher processor utilization may be hybrids between contiguous and scattered schemes. As the results of the research indicate, the negative effects of network contention due to job interactions will be increasingly visible in the future MPCs that can deliver messages as fast as the network capacity. Therefore, a goal for future research is to develop a hybrid processor allocation strategy that enjoys the increased processor utilizations as long as the negative effects of job interactions can be kept under control.

APPENDIX

APPENDIX

The following is given to complete Section 5.3.

Conditional Expectations

Let X be a continuous random variable having an exponential distribution with mean $\frac{1}{\lambda}$ and k be a constant. In this subsection we compute the expectation of X upon the condition that $X \geq k$, i.e. $E[X|X \geq k]$. The computation of $E[X|X < k]$ can be solved in a similar way as the computation of $E[X|X \geq k]$. To conserve space, we omit the computation of $E[X|X < k]$.

To compute the conditional expectation, $E[X|X \geq k]$, we have to solve the conditional probability density function ($p.d.f.$) conditioned by $X \geq k$.

$$\begin{aligned} f_{X|X\geq k}(x) &= 1 - Pr\{X \geq x | X \geq k\} = 1 - Pr\{X \geq x - k\} \\ &= \begin{cases} \lambda e^{-\lambda(x-k)}, & \text{if } x \geq k \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

And therefore the conditional expectation, $E[X|X \geq k]$ is

$$\begin{aligned} E[X|X \geq k] &= \int_k^\infty x\, f_{X|X\geq k}(x)\, dx \\ &= \int_k^\infty x\, \lambda e^{-\lambda(x-k)} = \lambda e^{\lambda k} \int_k^\infty x\, e^{-\lambda x}\, dx = k + \frac{1}{\lambda} \end{aligned}$$

Expectation of the Remaining Time, R_t

Consider a renewal process whose inter-arrival times, $T_0, T_1, ...$, are random variables having mean μ and variance σ. Although we do not need to specify T_i's distribution, for convenience we denote $F(x)$ and $f(x)$ as the *c.d.f.* and *p.d.f.* of T_i, respectively. Let R_t denote the time between the last renewal and the next renewal, and $G(x)$ and $g(x)$ be the *c.d.f.* and *p.d.f.* of R_t, respectively.

From renewal theory [103] we know that

$$\lim_{t\to\infty} Pr\{R_t < x\} = \frac{1}{\mu}\int_0^x (1 - F(y))\,dy$$

and therefore $g(x) = \frac{1}{\mu}(1 - F(x))$. Using this fact, we can compute the expectation of the R_t as follows:

$$\begin{aligned} E[R_t] &= \int_0^\infty x\, g(x)\,dx = \frac{1}{\mu}\int_0^\infty x\,(1 - F(x))\,dx \\ &= \frac{1}{2\mu}\int_0^\infty x^2\, f(x)\,dx = \frac{1}{2\mu}(\sigma^2 + \mu^2) \;=\; \frac{\mu}{2}(1 + \frac{\sigma^2}{\mu^2}) \end{aligned}$$

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] W. J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Transactions on Computers*, vol. 39, pp. 775–785, June 1990.

[2] Intel Corporation, *Paragon xp/s Product Overview*, 1991.

[3] W. J. Dally and C. L. Seitz, "The Torus Routing Chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.

[4] D. Min and M. W. Mutka, "Effects of Job Size Irregularity on the Dynamic Resource Scheduling of a 2-D Mesh Multicomputer," in *Proceedings of PARLE '93, Parallel Architectures and Languages Europe*, pp. 476–487, Jun. 1992.

[5] K. Li and K. H. Cheng, "A Two Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh Connected System," in *Proceedings 18th ACM Computer Science Conference*, pp. 22–28, Feb. 1990.

[6] P.-J. Chuang and N.-F. Tzeng, "An Efficient Submesh Allocation Strategy for Mesh Computer Systems," in *Proceedings of the 1991 International Conference on Distributed Computing Systems*, IEEE, May 1991.

[7] W. Liu, V. Lo, K. Windisch, and B. Nitzberg, "Non-contiguous Processor Allocation Algorithms for Distributed Memory Multicomputers," in *Proc. 1994 International Conference on Supercomputing*, pp. 227–236, Jun 1994.

[8] D. Min and M. W. Mutka, "A Multipath Contention Model for Analyzing Job Interactions in 2-D Mesh Multicomputers," in *Proceedings of the 1994 International Parallel Processing Symposium*, IEEE, Apr. 1994.

[9] D. Min and M. W. Mutka, "Determining External Contention Delay Due to Job Interactions in a 2-D Mesh Wormhole Routed Multicomputers," in *Proceedings of the 1993 Symposium on Parallel and Distributed Processing*, IEEE, Dec. 1993.

[10] D. Min and M. W. Mutka, "Effects of Job Interactions Due to Job Partitioning and Placement," Tech. Rep. CPS-95-1, Computer Science at Michigan State University, 1995.

[11] M. Chen and K. Shin, "Processor Allocation in an N-Cube Multiprocessor Using Gray Code," *IEEE Transactions on Computers*, vol. C-36, pp. 1396–1407, Dec. 1987.

[12] J. Kim, C. R. Das, and W. Lin, "A Top-Down Processor Allocation Scheme for Hypercube Computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, pp. 20–30, Jan. 1991.

[13] P.-J. Chuang and N.-F. Tzeng, "A Fast Recognition-Complete Processor Allocation Strategy for Hypercube Computers," *IEEE Transactions on Computers*, vol. 44, pp. 467–479, Apr. 1992.

[14] S. Dutt and J. P. Hayes, "Subcube Allocation in Hypercube Computers," *IEEE Transactions on Computers*, vol. 40, pp. 341–352, Mar. 1991.

[15] P. Krueger, T.-H. Lai, and V. A. Radiya, "Processor Allocation vs. Job Scheduling on Hypercube Computers," in *Proceedings of the 1991 International Conference on Distributed Computing Systems*, IEEE, May 1991.

[16] W. Crowther, J. Goodhue, E. Starr, R. Thomas, W. Millakem, and T. Blackadar, "Performance Measurements on a 128-Node Butterfly Parallel Processor," in *International Conference on Parallel Processing*, pp. 531–540, IEEE, Aug. 1985.

[17] T. Lovett and S. Thakkar, "The Symmetry Multiprocessor System," in *International Conference on Parallel Processing*, IEEE, 1988.

[18] Thacker and et al., "Firefly: A Multiprocessor Workstation," *IEEE Transactions on Computers*, pp. 909–920, Aug. 1988.

[19] R. Olson, "Parallel Processing in a Message-based Operating System," *IEEE Software*, vol. 2, pp. 39–49, July 1985.

[20] D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta, and J. Hennessy, "The DASH Prototype: Logic Overhead and Performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, pp. 41–61, Jan. 1993.

[21] Kendall Square Research, *KSR1 Principles of Operation, Revision 5.5*, 1991.

[22] Intel Corp., *Touchstone DELTA System Description*, 1991.

[23] W. D. Hillis and L. W. Tucker, "The CM-5 Connection Machine: A Scalable Supercomputer," *Communication of ACM*, vol. 36, pp. 31–40, Nov. 1993.

[24] N. Corp, *NCUBE/ten: An Overview.* Beaverton, OR, Nov. 1985.

[25] C. L. Seitz and et al., "The Architecture and Programming of the Amtek Series 2010 Multicomputer," in *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pp. 33–36, ACM, Jan. 1988.

[26] R. Alverson and et al., "The Tera Computer System," in *Proc. 1990 International Conference on Supercomputing*, pp. 1–6, June 1990.

[27] C. B. Stunkel, D. G. Shea, B. Abali, M. M. Dennean, P. H. Hochschild, D. J. Joseph, B. J. Nathanson, M. Tsao, and P. R. Varker, "Architecture and Implementation of Vulcan," in *Proceedings of the 8th International Parallel Processing Symposium*, pp. 268–274, IEEE, Apr. 1994.

[28] W. J. Dally, "The J-Machine: System Support for Actors," in *Actors: Knowledge-Based Concurrent Computing* (Hewitt and Agha, Eds.), MIT Press, 1989.

[29] B. Duzett and R. Buck, "An Overview of the nCUBE 3 Supercomputer," pp. 458–464, IEEE, Jul. 1992.

[30] The Cray Research, *Massively Parallel Processor System: CRAY T3D*, 1993.

[31] N. Koike, "NEC Cenju-3: A Multiprocessor-Based Parallel Computer," in *Proceedings of the 1994 International Parallel Processing Symposium*, pp. 396–401, IEEE, Apr. 1994.

[32] J. Zahorjan and C. McCann, "Processor Scheduling in Shared Memory Multiprocessors," in *Proceedings of the 1990 ACM Conference on Measurement and Modeling of Computer Systems*, pp. 214–225, ACM, May 1990.

[33] S. Majumdar, D. L. Eager, and R. B. Bunt, "Scheduling in Multiprogrammed Parallel Systems," in *Proceedings of the 1988 Conference on Measurement and Modeling of Computer Systems*, pp. 104–113, ACM, May. 1988.

[34] S. T. Leutenegger and M. K. Vernon, "The Performance of Multiprogrammed Multiprocessor Scheduling Algorithms," in *Proceedings of the 1990 ACM Conference on Measurement and Modeling of Computer Systems*, pp. 226–236, ACM, May 1990.

[35] A. Tucker and A. Gupta, "Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors," in *Proceedings 12th ACM Symposium on Operating System Principles*, ACM, Nov. 1989.

[36] K. Li and K. H. Cheng, "Static Job Scheduling in Partitonable Mesh Connected Systems," *Journal of Parallel and Distributed Computing*, pp. 152–159, Oct. 1990.

[37] H. El-Rewini and T. G. Lewis, "Scheduling Parallel Program Tasks onto Arbitrary Target Machines," *Journal of Parallel and Distributed Computing*, pp. 138–153, Sep. 1990.

[38] F. Ercal, J. Ramanujam, and P. Sadayappan, "Task Allocation onto a Hypercube by Recursive Mincut Bipartitioning," *Journal of Parallel and Distributed Computing*, pp. 35–44, Oct. 1990.

[39] B. Shirazi and M. Wang, "Analysis and Evaluation of Heuristic Methods for Static Task Scheduling," *Journal of Parallel and Distributed Computing*, pp. 222–232, Oct. 1990.

[40] Y. Zhu and M. Ahuja, "Job Scheduling on a Hypercube," in *Proceedings of the 1990 Conference on Distributed Computing Systems*, pp. 510–517, IEEE, Jun. 1990.

[41] D.-T. Peng and K. G. Shin, "Static Allocation of Periodic Tasks with Precedence Constraints in Distributed Real-Time Systems," in *Proc. of Intl. Conf. on Distributed Computing Systems*, pp. 190–198, IEEE, 1990.

[42] K. Ramamritham, "Allocation and Scheduling of Complex Periodic Tasks," in *Proc. of 1990 Int. Conference on Distributed Computing Systems*, pp. 108–115, IEEE, 1990.

[43] Q. Wang and K. H. Cheng, "A Heuristic of Scheduling Parallel Tasks and its Analysis," *SIAM Journal on Computing*, vol. 21, pp. 281–294, Apr. 1992.

[44] J. Blazewicz, M. Drabowski, and J. Weglarz, "Scheduling Multiprocessor Tasks to Minimize Schedule Length," *IEEE Transactions on Computers*, vol. 35, pp. 389–393, May 1986.

[45] C. Coroyer and Z. Liu, "Effectiveness of Heuristics and Simulated Annealing for the Scheduling of Concurrent Tasks – An Empirical Comparison," in *Proceedings of PARLE '93, Parallel Architectures and Languages Europe*, Jun. 1993.

[46] M. R. Garey and R. L. Graham, "Bounds For Multiprocessor Scheduling with Resource Constraints," *SIAM Journal on Computing*, vol. 4, pp. 187–200, Jun. 1975.

[47] J. Du and J. Y.-T. Leung, "Complexity of Scheduling Parallel Task Systems," *SIAM Journal on Computing*, vol. 2, pp. 473–487, Nov. 1989.

[48] S. Y. Lee and J. K. Aggarwal, "A Mapping Strategy for Parallel Processing," *IEEE Transactions on Computers*, pp. 433–442, Apr. 1987.

[49] S. H. Bokhari, "On the Mapping Problem," *IEEE Transactions on Computers*, pp. 207–214, Mar. 1981.

[50] P. Sadayappan and F. Ercal, "Nearest-Neighbor Mapping of Finite Element Graphs onto Processor Meshes," *IEEE Transactions on Computers*, pp. 1408–1424, Dec. 1987.

[51] C. T. Ho and S. L. Johnsson, "On the Embedding of Arbitrary Meshes in Boolean Cubes with Expansion Two Dilation Two," in *International Conference on Parallel Processing*, pp. 188–191, IEEE, 1987.

[52] M. Y. Chan, "Embedding of Grids into Optiaml Hypercubes," *SIAM Journal on Computing*, vol. 20, pp. 834–864, Oct. 1991.

[53] R. Aleliunas and A. L. Rosenberg, "On Embedding Rectangular Grids in Square Grids," *IEEE Transactions on Computers*, pp. 907–913, Sep. 1982.

[54] J. A. Ellis, "Embedding Rectangular Grids into Square Grids," *IEEE Transactions on Computers*, pp. 46–52, Jan. 1991.

[55] R. G. Melhem, "Embedding Rectangular Grids into Square Grids with Dilation Two," *IEEE Transactions on Computers*, pp. 1446–1455, Dec. 1990.

[56] Y. E. Ma and L. Tao, "Embeddings Among Toruses and Meshes," in *International Conference on Parallel Processing*, pp. 178–187, IEEE, 1987.

[57] K. Efe, "Embedding Mesh of Trees in the Hypercube," *Journal of Parallel and Distributed Computing*, pp. 222–230, 1991.

[58] A. Y. Wu, "Embedding of Tree Networks into Hypercubes," *Journal of Parallel and Distributed Computing*, pp. 238–249, 1985.

[59] R. Varadarajan, "Embedding Shuffle Networks in Hypercubes," *Journal of Parallel and Distributed Computing*, pp. 252–256, 1991.

[60] K. Knowlton, "A Fast Storage Allocator," *Communication of ACM*, vol. 8, pp. 623–625, Oct. 1965.

[61] D. D. Sharma and D. K. Pradhan, "Fast and Efficient Strategies for Cubic and Non-cubic Allocation in Hypercube Multiprocessors," in *International Conference on Parallel Processing*, vol. I, pp. 118–127, IEEE, 1993.

[62] H. Wang and Q. Yang, "Prime Cube Graph Approach for Processor Allocation in Hypercube Multiprocessors," in *International Conference on Parallel Processing*, vol. I, pp. 25–32, IEEE, 1991.

[63] P. Mohapatra, C.Yu, C.R.Das, and J.Kim, "A Lazy Scheduling Scheme for Improving Hypercube Performance," in *International Conference on Parallel Processing*, vol. I, pp. 110–117, IEEE, 1993.

[64] M.-S. Chen and K. G. Shin, "Subcube Allocation and Task Migration in Hypercube Multiprocessors," *IEEE Transactions on Computers*, vol. 39, pp. 1146–1155, Sep. 1990.

[65] C. Yu and C. R. Das, "Limit Allocation: An Efficient Processor Management Scheme for Hypercubes," in *International Conference on Parallel Processing*, vol. II, pp. 143–150, IEEE, 1994.

[66] K. Li and K.-H. Cheng, "Job Scheduling in a Parallel Mesh Using a Two-Dimensional Buddy System Partitioning Scheme," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, pp. 413–422, Oct. 1991.

[67] J. Ding and L. N. Bhuyan, "An Adaptive Submesh Allocation Strategies for Two-Dimensional Mesh Connected Systems," in *International Conference on Parallel Processing*, vol. II, pp. 193–200, IEEE, 1993.

[68] Y. Zhu, "Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers," *Journal of Parallel and Distributed Computing*, pp. 328–337, 1992.

[69] D. D. Sharma and D. K. Pradhan, "A Fast and Efficient Strategy for Submesh Allocation in Mesh-Connected Parallel Computer," in *Proceedings of the 1993 Symposium on Parallel and Distributed Processing*, pp. 682–689, IEEE, Dec. 1993.

[70] D. D. Sharma and D. K. Pradhan, "Job Scheduling in Mesh Multicomputers," in *International Conference on Parallel Processing*, vol. II, pp. 251–258, IEEE, 1994.

[71] S. Bhattacharya and W.-T. Tsai, "Lookahead Processor Allocation in Mesh-Connected Massively Parallel Multicomputer," in *Proceedings of the 1994 International Parallel Processing Symposium*, pp. 868–875, IEEE, Apr. 1994.

[72] M. R. Garey and D. S. Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness.* Freeman, 1988.

[73] B. Baker, E. Coffman, and R. Rivest, "Orthogonal Packings in Two Dimensions," *SIAM Journal on Computing*, vol. 4, pp. 846–855, Nov. 1980.

[74] E. Coffman, J. F. M.R., J. D.S., and R. Tarjan, "Performance Bounds for Level-oriented Two-dimensional Packing Algorithms," *SIAM Journal on Computing*, vol. 4, pp. 808–826, Nov. 1980.

[75] E. Coffman, J. Leung, J.Y., and D. Slutz, "On the Optinality of First-Fit and Level Algorithms for Parallel Machine Assignments and Sequencing," in *1977 International Conference on Parallel Processing*, pp. 95–99, IEEE, 1977.

[76] K. Li and K. H. Cheng, "Static Job Scheduling in Partitionable Mesh Connected Systems," *Journal of Parallel and Distributed Computing*, pp. 152–159, Oct. 1990.

[77] J. Turek, J. L. Wolf, and P. S. Yu, "Approximate Algorithms for Parallelizable Tasks," in *Symposium on Parallel Architecture and Language*, pp. 323–332, IEEE, 1992.

[78] K. Li and K. Cheng, "Complexitiy of Resource Allocation and Job Scheduling Problems in Partitionable Mesh Connected Systems," in *Proceedings of the 1st Annual IEEE Symposium on Parallel and Distributed Processing*, IEEE, May 1989.

[79] J. Y.-T. Leung, T. Tam, C. Wong, G. Young, and F. Chin, "Packing Squares into a Squre," Tech. Rep. UTDCS-12-89, Computer Science Department, University of Texas at Dallas, July 1989.

[80] S. Chittor and R. Enbody, "Hypercube vs. 2d meshes," in *Proceedings SIAM Fourth Annual Conference on Parallel Processing for Scientific Computing*, pp. 313–318, SIAM, 1990.

[81] S. Chittor and R. Enbody, "Predicting The Effect of Mapping on The Communication Performance of Large Multicomputers," in *1991 International Conference on Parallel Processing*, IEEE, 1991.

[82] S. Chittor and R. Enbody, "High-Performance Simulator for Large Wormhole-Routed Networks," *International Journal in Computer Simulator*, pp. 151–174, 1992.

[83] V. S. Adve and M. K. Vernon, "Performance Analysis of Multiprocessor Mesh Interconnection Networks with Wormhole Routing," tech. rep., Department of Computer Science at University of Wisconsin-Madison, June 1992.

[84] S. Abraham, "Performance of Multicomputer Networks under Pin-out Constraints," *Journal of Parallel and Distributed Computing*, pp. 237–248, 1991.

[85] J. H. Kim and A. A. Chien, "The Impact of Packetization in Wormhole-Routed Networks," in *Proceedings of PARLE '93, Parallel Architectures and Languages Europe*, pp. 242–253, Jun. 1993.

[86] W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, pp. 547–553, May 1987.

[87] W. J. Dally, "Express Cubes: Improving the Performance of k-ary n-cube Interconnection Networks," *IEEE Transactions on Computers*, vol. 40, pp. 1016–1023, Sep. 1991.

[88] W. J. Dally, "Virtual-Channel Flow Control," *Journal of Parallel and Distributed Computing*, pp. 194–205, 1992.

[89] L. M. Ni and P. K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *IEEE Computer*, vol. 26, pp. 62 – 76, Feb. 1993.

[90] C. J. Glass and L. M. Ni, "The Turn Model for Adaptive Routing," in *Proceedings of the 19th Annual International Symposium on Computer Architecture*, May 1992.

[91] Z. Liu, J. Duato, and L.-E. Thorelli, "Grouping Virtual Channels for Deadlock-Free Adaptive Wormhole Routing," in *Proceedings of PARLE '93, Parallel Architectures and Languages Europe*, pp. 254–265, Jun. 1993.

[92] D. H. Linder and J. C. Harden, "An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes," *IEEE Transactions on Computers*, pp. 2–12, Jan. 1991.

[93] K. J. H. and A. A. Chien, "Evaluation of Wormhole Routed Networks Under Hybrid Traffic Loads," in *Proceedings of the Hawaii International Conference on System Sciences*, Jan 1993.

[94] S. Borkar and et al., "An Integrated Solution to High-Speed Parallel Computing," in *Proc. of the Supercomputing Conference*, pp. 330–339, 1988.

[95] P. Y. Song, "Design of a Network for Concurrent Message Passing Systems," Tech. Rep. Master's Thesis, Department of Electrical Engineering at MIT, May 1988.

[96] S. F. P. Raghavan and E. Upfal, "A Theory of Wormhole Routing in Parallel Computers," in *Symposium on the Foundations of Computer Science*, pp. 563–572, 1992.

[97] E. A. Brewer and B. C. Kuszmaul, "How to Get Good Performance from the CM-5 Data Network," in *Proceedings of the 1994 International Parallel Processing Sympos ium*, pp. 858 - 867, IEEE, Apr. 1994.

[98] P. K. McKinley, "Simulation of Wormhole Routing in Multisim," in *Proc. 23rd Pittsburg Conference on Modeling and Simulation*, Apr. 1992.

[99] H. Schwetman, "CSIM: C-Based, Process-Oriented Simulation Language," Tech. Rep. PP-080-85, Microelectronics and Computer Technology Corporation, 1985.

[100] L. Kleinrock, *Queueing Systems, Volume I: Theory.* John Wiley and Sons, 1975.

[101] S. Chittor and R. Enbody, "Performance Evaluation of Mesh-connected Wormhole-routed Networks for Interprocessor Communication in Multicomputers," in *Proceedings of Supercomputing Conference*, pp. 647–656, Nov. 1990.

[102] D. Min and M. W. Mutka, "A Framework for Predicting Delay Due to Job Interactions in a 2-D Mesh Multicomputer," in *Proceedings of the 1993 International Parallel Processing Symposium*, pp. 350 - 357, IEEE, Apr. 1993.

[103] S. M. Ross, *Stochastic Processes.* John Wiley and Sons, 1983.

[104] F. A. G. Alexander M. Mood and D. C. Boes, *Introduction to the Theory of Statistics.* McGraw Hill, 1974.

APPENDIX

The following is given to complete Section 5.3.

Conditional Expectations

Let X be a continuous random variable having an exponential distribution with mean $\frac{1}{\lambda}$ and k be a constant. In this subsection we compute the expectation of X upon the condition that $X \geq k$, i.e. $E[X|X \geq k]$. The computation of $E[X|X < k]$ can be solved in a similar way as the computation of $E[X|X \geq k]$. To conserve space, we omit the computation of $E[X|X < k]$.

To compute the conditional expectation, $E[X|X \geq k]$, we have to solve the conditional probability density function ($p.d.f.$) conditioned by $X \geq k$.

$$\begin{aligned} f_{X|X\geq k}(x) &= 1 - Pr\{X \geq x | X \geq k\} = 1 - Pr\{X \geq x - k\} \\ &= \begin{cases} \lambda e^{-\lambda(x-k)}, & \text{if } x \geq k \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

And therefore the conditional expectation, $E[X|X \geq k]$ is

$$\begin{aligned} E[X|X \geq k] &= \int_k^\infty x\, f_{X|X\geq k}(x)\, dx \\ &= \int_k^\infty x\, \lambda e^{-\lambda(x-k)} = \lambda e^{\lambda k} \int_k^\infty x\, e^{-\lambda x}\, dx = k + \frac{1}{\lambda} \end{aligned}$$

Expectation of the Remaining Time, R_t

Consider a renewal process whose inter-arrival times, $T_0, T_1, ...$, are random variables having mean μ and variance σ. Although we do not need to specify T_i's distribution, for convenience we denote $F(x)$ and $f(x)$ as the *c.d.f.* and *p.d.f.* of T_i, respectively. Let R_t denote the time between the last renewal and the next renewal, and $G(x)$ and $g(x)$ be the *c.d.f.* and *p.d.f.* of R_t, respectively.

From renewal theory [103] we know that

$$\lim_{t\to\infty} Pr\{R_t < x\} = \frac{1}{\mu}\int_0^x (1 - F(y))\,dy$$

and therefore $g(x) = \frac{1}{\mu}(1 - F(x))$. Using this fact, we can compute the expectation of the R_t as follows:

$$\begin{aligned} E[R_t] &= \int_0^\infty x\, g(x)\,dx = \frac{1}{\mu}\int_0^\infty x\,(1 - F(x))\,dx \\ &= \frac{1}{2\mu}\int_0^\infty x^2\, f(x)\,dx = \frac{1}{2\mu}(\sigma^2 + \mu^2) \;=\; \frac{\mu}{2}(1 + \frac{\sigma^2}{\mu^2}) \end{aligned}$$

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] W. J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Transactions on Computers*, vol. 39, pp. 775–785, June 1990.

[2] Intel Corporation, *Paragon xp/s Product Overview*, 1991.

[3] W. J. Dally and C. L. Seitz, "The Torus Routing Chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.

[4] D. Min and M. W. Mutka, "Effects of Job Size Irregularity on the Dynamic Resource Scheduling of a 2-D Mesh Multicomputer," in *Proceedings of PARLE '93, Parallel Architectures and Languages Europe*, pp. 476–487, Jun. 1992.

[5] K. Li and K. H. Cheng, "A Two Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh Connected System," in *Proceedings 18th ACM Computer Science Conference*, pp. 22–28, Feb. 1990.

[6] P.-J. Chuang and N.-F. Tzeng, "An Efficient Submesh Allocation Strategy for Mesh Computer Systems," in *Proceedings of the 1991 International Conference on Distributed Computing Systems*, IEEE, May 1991.

[7] W. Liu, V. Lo, K. Windisch, and B. Nitzberg, "Non-contiguous Processor Allocation Algorithms for Distributed Memory Multicomputers," in *Proc. 1994 International Conference on Supercomputing*, pp. 227–236, Jun 1994.

[8] D. Min and M. W. Mutka, "A Multipath Contention Model for Analyzing Job Interactions in 2-D Mesh Multicomputers," in *Proceedings of the 1994 International Parallel Processing Symposium*, IEEE, Apr. 1994.

[9] D. Min and M. W. Mutka, "Determining External Contention Delay Due to Job Interactions in a 2-D Mesh Wormhole Routed Multicomputers," in *Proceedings of the 1993 Symposium on Parallel and Distributed Processing*, IEEE, Dec. 1993.

[10] D. Min and M. W. Mutka, "Effects of Job Interactions Due to Job Partitioning and Placement," Tech. Rep. CPS-95-1, Computer Science at Michigan State University, 1995.

[11] M. Chen and K. Shin, "Processor Allocation in an N-Cube Multiprocessor Using Gray Code," *IEEE Transactions on Computers*, vol. C-36, pp. 1396–1407, Dec. 1987.

[12] J. Kim, C. R. Das, and W. Lin, "A Top-Down Processor Allocation Scheme for Hypercube Computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, pp. 20–30, Jan. 1991.

[13] P.-J. Chuang and N.-F. Tzeng, "A Fast Recognition-Complete Processor Allocation Strategy for Hypercube Computers," *IEEE Transactions on Computers*, vol. 44, pp. 467–479, Apr. 1992.

[14] S. Dutt and J. P. Hayes, "Subcube Allocation in Hypercube Computers," *IEEE Transactions on Computers*, vol. 40, pp. 341–352, Mar. 1991.

[15] P. Krueger, T.-H. Lai, and V. A. Radiya, "Processor Allocation vs. Job Scheduling on Hypercube Computers," in *Proceedings of the 1991 International Conference on Distributed Computing Systems*, IEEE, May 1991.

[16] W. Crowther, J. Goodhue, E. Starr, R. Thomas, W. Millakem, and T. Blackadar, "Performance Measurements on a 128-Node Butterfly Parallel Processor," in *International Conference on Parallel Processing*, pp. 531–540, IEEE, Aug. 1985.

[17] T. Lovett and S. Thakkar, "The Symmetry Multiprocessor System," in *International Conference on Parallel Processing*, IEEE, 1988.

[18] Thacker and et al., "Firefly: A Multiprocessor Workstation," *IEEE Transactions on Computers*, pp. 909–920, Aug. 1988.

[19] R. Olson, "Parallel Processing in a Message-based Operating System," *IEEE Software*, vol. 2, pp. 39–49, July 1985.

[20] D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta, and J. Hennessy, "The DASH Prototype: Logic Overhead and Performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, pp. 41–61, Jan. 1993.

[21] Kendall Square Research, *KSR1 Principles of Operation, Revision 5.5*, 1991.

[22] Intel Corp., *Touchstone DELTA System Description*, 1991.

[23] W. D. Hillis and L. W. Tucker, "The CM-5 Connection Machine: A Scalable Supercomputer," *Communication of ACM*, vol. 36, pp. 31–40, Nov. 1993.

[24] N. Corp, *NCUBE/ten: An Overview.* Beaverton, OR, Nov. 1985.

[25] C. L. Seitz and et al., "The Architecture and Programming of the Amtek Series 2010 Multicomputer," in *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pp. 33–36, ACM, Jan. 1988.

[26] R. Alverson and et al., "The Tera Computer System," in *Proc. 1990 International Conference on Supercomputing*, pp. 1–6, June 1990.

[27] C. B. Stunkel, D. G. Shea, B. Abali, M. M. Dennean, P. H. Hochschild, D. J. Joseph, B. J. Nathanson, M. Tsao, and P. R. Varker, "Architecture and Implementation of Vulcan," in *Proceedings of the 8th International Parallel Processing Symposium*, pp. 268–274, IEEE, Apr. 1994.

[28] W. J. Dally, "The J-Machine: System Support for Actors," in *Actors: Knowledge-Based Concurrent Computing* (Hewitt and Agha, Eds.), MIT Press, 1989.

[29] B. Duzett and R. Buck, "An Overview of the nCUBE 3 Supercomputer," pp. 458–464, IEEE, Jul. 1992.

[30] The Cray Research, *Massively Parallel Processor System: CRAY T3D*, 1993.

[31] N. Koike, "NEC Cenju-3: A Multiprocessor-Based Parallel Computer," in *Proceedings of the 1994 International Parallel Processing Symposium*, pp. 396–401, IEEE, Apr. 1994.

[32] J. Zahorjan and C. McCann, "Processor Scheduling in Shared Memory Multiprocessors," in *Proceedings of the 1990 ACM Conference on Measurement and Modeling of Computer Systems*, pp. 214–225, ACM, May 1990.

[33] S. Majumdar, D. L. Eager, and R. B. Bunt, "Scheduling in Multiprogrammed Parallel Systems," in *Proceedings of the 1988 Conference on Measurement and Modeling of Computer Systems*, pp. 104–113, ACM, May. 1988.

[34] S. T. Leutenegger and M. K. Vernon, "The Performance of Multiprogrammed Multiprocessor Scheduling Algorithms," in *Proceedings of the 1990 ACM Conference on Measurement and Modeling of Computer Systems*, pp. 226–236, ACM, May 1990.

[35] A. Tucker and A. Gupta, "Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors," in *Proceedings 12th ACM Symposium on Operating System Principles*, ACM, Nov. 1989.

[36] K. Li and K. H. Cheng, "Static Job Scheduling in Partitonable Mesh Connected Systems," *Journal of Parallel and Distributed Computing*, pp. 152–159, Oct. 1990.

[37] H. El-Rewini and T. G. Lewis, "Scheduling Parallel Program Tasks onto Arbitrary Target Machines," *Journal of Parallel and Distributed Computing*, pp. 138–153, Sep. 1990.

[38] F. Ercal, J. Ramanujam, and P. Sadayappan, "Task Allocation onto a Hypercube by Recursive Mincut Bipartitioning," *Journal of Parallel and Distributed Computing*, pp. 35–44, Oct. 1990.

[39] B. Shirazi and M. Wang, "Analysis and Evaluation of Heuristic Methods for Static Task Scheduling," *Journal of Parallel and Distributed Computing*, pp. 222–232, Oct. 1990.

[40] Y. Zhu and M. Ahuja, "Job Scheduling on a Hypercube," in *Proceedings of the 1990 Conference on Distributed Computing Systems*, pp. 510–517, IEEE, Jun. 1990.

[41] D.-T. Peng and K. G. Shin, "Static Allocation of Periodic Tasks with Precedence Constraints in Distributed Real-Time Systems," in *Proc. of Intl. Conf. on Distributed Computing Systems*, pp. 190–198, IEEE, 1990.

[42] K. Ramamritham, "Allocation and Scheduling of Complex Periodic Tasks," in *Proc. of 1990 Int. Conference on Distributed Computing Systems*, pp. 108–115, IEEE, 1990.

[43] Q. Wang and K. H. Cheng, "A Heuristic of Scheduling Parallel Tasks and its Analysis," *SIAM Journal on Computing*, vol. 21, pp. 281–294, Apr. 1992.

[44] J. Blazewicz, M. Drabowski, and J. Weglarz, "Scheduling Multiprocessor Tasks to Minimize Schedule Length," *IEEE Transactions on Computers*, vol. 35, pp. 389–393, May 1986.

[45] C. Coroyer and Z. Liu, "Effectiveness of Heuristics and Simulated Annealing for the Scheduling of Concurrent Tasks – An Empirical Comparison," in *Proceedings of PARLE '93, Parallel Architectures and Languages Europe*, Jun. 1993.

[46] M. R. Garey and R. L. Graham, "Bounds For Multiprocessor Scheduling with Resource Constraints," *SIAM Journal on Computing*, vol. 4, pp. 187–200, Jun. 1975.

[47] J. Du and J. Y.-T. Leung, "Complexity of Scheduling Parallel Task Systems," *SIAM Journal on Computing*, vol. 2, pp. 473–487, Nov. 1989.

[48] S. Y. Lee and J. K. Aggarwal, "A Mapping Strategy for Parallel Processing," *IEEE Transactions on Computers*, pp. 433–442, Apr. 1987.

[49] S. H. Bokhari, "On the Mapping Problem," *IEEE Transactions on Computers*, pp. 207–214, Mar. 1981.

[50] P. Sadayappan and F. Ercal, "Nearest-Neighbor Mapping of Finite Element Graphs onto Processor Meshes," *IEEE Transactions on Computers*, pp. 1408–1424, Dec. 1987.

[51] C. T. Ho and S. L. Johnsson, "On the Embedding of Arbitrary Meshes in Boolean Cubes with Expansion Two Dilation Two," in *International Conference on Parallel Processing*, pp. 188–191, IEEE, 1987.

[52] M. Y. Chan, "Embedding of Grids into Optiaml Hypercubes," *SIAM Journal on Computing*, vol. 20, pp. 834–864, Oct. 1991.

[53] R. Aleliunas and A. L. Rosenberg, "On Embedding Rectangular Grids in Square Grids," *IEEE Transactions on Computers*, pp. 907–913, Sep. 1982.

[54] J. A. Ellis, "Embedding Rectangular Grids into Square Grids," *IEEE Transactions on Computers*, pp. 46–52, Jan. 1991.

[55] R. G. Melhem, "Embedding Rectangular Grids into Square Grids with Dilation Two," *IEEE Transactions on Computers*, pp. 1446–1455, Dec. 1990.

[56] Y. E. Ma and L. Tao, "Embeddings Among Toruses and Meshes," in *International Conference on Parallel Processing*, pp. 178–187, IEEE, 1987.

[57] K. Efe, "Embedding Mesh of Trees in the Hypercube," *Journal of Parallel and Distributed Computing*, pp. 222–230, 1991.

[58] A. Y. Wu, "Embedding of Tree Networks into Hypercubes," *Journal of Parallel and Distributed Computing*, pp. 238–249, 1985.

[59] R. Varadarajan, "Embedding Shuffle Networks in Hypercubes," *Journal of Parallel and Distributed Computing*, pp. 252–256, 1991.

[60] K. Knowlton, "A Fast Storage Allocator," *Communication of ACM*, vol. 8, pp. 623–625, Oct. 1965.

[61] D. D. Sharma and D. K. Pradhan, "Fast and Efficient Strategies for Cubic and Non-cubic Allocation in Hypercube Multiprocessors," in *International Conference on Parallel Processing*, vol. I, pp. 118–127, IEEE, 1993.

[62] H. Wang and Q. Yang, "Prime Cube Graph Approach for Processor Allocation in Hypercube Multiprocessors," in *International Conference on Parallel Processing*, vol. I, pp. 25–32, IEEE, 1991.

[63] P. Mohapatra, C.Yu, C.R.Das, and J.Kim, "A Lazy Scheduling Scheme for Improving Hypercube Performance," in *International Conference on Parallel Processing*, vol. I, pp. 110–117, IEEE, 1993.

[64] M.-S. Chen and K. G. Shin, "Subcube Allocation and Task Migration in Hypercube Multiprocessors," *IEEE Transactions on Computers*, vol. 39, pp. 1146–1155, Sep. 1990.

[65] C. Yu and C. R. Das, "Limit Allocation: An Efficient Processor Management Scheme for Hypercubes," in *International Conference on Parallel Processing*, vol. II, pp. 143–150, IEEE, 1994.

[66] K. Li and K.-H. Cheng, "Job Scheduling in a Parallel Mesh Using a Two-Dimensional Buddy System Partitioning Scheme," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, pp. 413–422, Oct. 1991.

[67] J. Ding and L. N. Bhuyan, "An Adaptive Submesh Allocation Strategies for Two-Dimensional Mesh Connected Systems," in *International Conference on Parallel Processing*, vol. II, pp. 193–200, IEEE, 1993.

[68] Y. Zhu, "Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers," *Journal of Parallel and Distributed Computing*, pp. 328–337, 1992.

[69] D. D. Sharma and D. K. Pradhan, "A Fast and Efficient Strategy for Submesh Allocation in Mesh-Connected Parallel Computer," in *Proceedings of the 1993 Symposium on Parallel and Distributed Processing*, pp. 682–689, IEEE, Dec. 1993.

[70] D. D. Sharma and D. K. Pradhan, "Job Scheduling in Mesh Multicomputers," in *International Conference on Parallel Processing*, vol. II, pp. 251–258, IEEE, 1994.

[71] S. Bhattacharya and W.-T. Tsai, "Lookahead Processor Allocation in Mesh-Connected Massively Parallel Multicomputer," in *Proceedings of the 1994 International Parallel Processing Symposium*, pp. 868–875, IEEE, Apr. 1994.

[72] M. R. Garey and D. S. Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness.* Freeman, 1988.

[73] B. Baker, E. Coffman, and R. Rivest, "Orthogonal Packings in Two Dimensions," *SIAM Journal on Computing*, vol. 4, pp. 846–855, Nov. 1980.

[74] E. Coffman, J. F. M.R., J. D.S., and R. Tarjan, "Performance Bounds for Level-oriented Two-dimensional Packing Algorithms," *SIAM Journal on Computing*, vol. 4, pp. 808–826, Nov. 1980.

[75] E. Coffman, J. Leung, J.Y., and D. Slutz, "On the Optinality of First-Fit and Level Algorithms for Parallel Machine Assignments and Sequencing," in *1977 International Conference on Parallel Processing*, pp. 95–99, IEEE, 1977.

[76] K. Li and K. H. Cheng, "Static Job Scheduling in Partitionable Mesh Connected Systems," *Journal of Parallel and Distributed Computing*, pp. 152–159, Oct. 1990.

[77] J. Turek, J. L. Wolf, and P. S. Yu, "Approximate Algorithms for Parallelizable Tasks," in *Symposium on Parallel Architecture and Language*, pp. 323–332, IEEE, 1992.

[78] K. Li and K. Cheng, "Complexitiy of Resource Allocation and Job Scheduling Problems in Partitionable Mesh Connected Systems," in *Proceedings of the 1st Annual IEEE Symposium on Parallel and Distributed Processing*, IEEE, May 1989.

[79] J. Y.-T. Leung, T. Tam, C. Wong, G. Young, and F. Chin, "Packing Squares into a Squre," Tech. Rep. UTDCS-12-89, Computer Science Department, University of Texas at Dallas, July 1989.

[80] S. Chittor and R. Enbody, "Hypercube vs. 2d meshes," in *Proceedings SIAM Fourth Annual Conference on Parallel Processing for Scientific Computing*, pp. 313–318, SIAM, 1990.

[81] S. Chittor and R. Enbody, "Predicting The Effect of Mapping on The Communication Performance of Large Multicomputers," in *1991 International Conference on Parallel Processing*, IEEE, 1991.

[82] S. Chittor and R. Enbody, "High-Performance Simulator for Large Wormhole-Routed Networks," *International Journal in Computer Simulator*, pp. 151–174, 1992.

[83] V. S. Adve and M. K. Vernon, "Performance Analysis of Multiprocessor Mesh Interconnection Networks with Wormhole Routing," tech. rep., Department of Computer Science at University of Wisconsin-Madison, June 1992.

[84] S. Abraham, "Performance of Multicomputer Networks under Pin-out Constraints," *Journal of Parallel and Distributed Computing*, pp. 237–248, 1991.

[85] J. H. Kim and A. A. Chien, "The Impact of Packetization in Wormhole-Routed Networks," in *Proceedings of PARLE '93, Parallel Architectures and Languages Europe*, pp. 242–253, Jun. 1993.

[86] W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, pp. 547–553, May 1987.

[87] W. J. Dally, "Express Cubes: Improving the Performance of k-ary n-cube Interconnection Networks," *IEEE Transactions on Computers*, vol. 40, pp. 1016–1023, Sep. 1991.

[88] W. J. Dally, "Virtual-Channel Flow Control," *Journal of Parallel and Distributed Computing*, pp. 194–205, 1992.

[89] L. M. Ni and P. K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *IEEE Computer*, vol. 26, pp. 62 – 76, Feb. 1993.

[90] C. J. Glass and L. M. Ni, "The Turn Model for Adaptive Routing," in *Proceedings of the 19th Annual International Symposium on Computer Architecture*, May 1992.

[91] Z. Liu, J. Duato, and L.-E. Thorelli, "Grouping Virtual Channels for Deadlock-Free Adaptive Wormhole Routing," in *Proceedings of PARLE '93, Parallel Architectures and Languages Europe*, pp. 254–265, Jun. 1993.

[92] D. H. Linder and J. C. Harden, "An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes," *IEEE Transactions on Computers*, pp. 2–12, Jan. 1991.

[93] K. J. H. and A. A. Chien, "Evaluation of Wormhole Routed Networks Under Hybrid Traffic Loads," in *Proceedings of the Hawaii International Conference on System Sciences*, Jan 1993.

[94] S. Borkar and et al., "An Integrated Solution to High-Speed Parallel Computing," in *Proc. of the Supercomputing Conference*, pp. 330–339, 1988.

[95] P. Y. Song, "Design of a Network for Concurrent Message Passing Systems," Tech. Rep. Master's Thesis, Department of Electrical Engineering at MIT, May 1988.

[96] S. F. P. Raghavan and E. Upfal, "A Theory of Wormhole Routing in Parallel Computers," in *Symposium on the Foundations of Computer Science*, pp. 563–572, 1992.

[97] E. A. Brewer and B. C. Kuszmaul, "How to Get Good Performance from the CM-5 Data Network," in *Proceedings of the 1994 International Parallel Processing Sympos ium*, pp. 858 – 867, IEEE, Apr. 1994.

[98] P. K. McKinley, "Simulation of Wormhole Routing in Multisim," in *Proc. 23rd Pittsburg Conference on Modeling and Simulation*, Apr. 1992.

[99] H. Schwetman, "CSIM: C-Based, Process-Oriented Simulation Language," Tech. Rep. PP-080-85, Microelectronics and Computer Technology Corporation, 1985.

[100] L. Kleinrock, *Queueing Systems, Volume I: Theory.* John Wiley and Sons, 1975.

[101] S. Chittor and R. Enbody, "Performance Evaluation of Mesh-connected Wormhole-routed Networks for Interprocessor Communication in Multicomputers," in *Proceedings of Supercomputing Conference*, pp. 647–656, Nov. 1990.

[102] D. Min and M. W. Mutka, "A Framework for Predicting Delay Due to Job Interactions in a 2-D Mesh Multicomputer," in *Proceedings of the 1993 International Parallel Processing Symposium*, pp. 350 – 357, IEEE, Apr. 1993.

[103] S. M. Ross, *Stochastic Processes.* John Wiley and Sons, 1983.

[104] F. A. G. Alexander M. Mood and D. C. Boes, *Introduction to the Theory of Statistics.* McGraw Hill, 1974.

[105] P. K. McKinley and C. Trefftz, "MultiSim: A Tool for the Study of Large-scale Multiprocessors," in *Proc. 1993 International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Networks (MASCOTS 93)*, pp. 57–62, Jan 1993.

[106] A. Silberschatz, J. Peterson, and P. Galvin, *Operating System Concepts.* Addison Wesley, 3rd ed., 1991.